Arquitetura de Computadores

Sumário

Por Matéria

Por Aula

- Aula 4 - 17/04/2015
- Aula 5 - 24/04/2015
- Aula 6 - 08/05/2015
- Aula 7 - 15/05/2015
- **Prova** - 22/05/2015
- [Aula 8]

Aula 1 - 20/03/2015

Sinais

Noções de Sinais e Processamento por Computadores

Sinal é uma entidade física que descreve matematicamente a emissão de algum tipo de energia e como ela varia ao longo do tempo.

> **Ex:** Sinal de áudio, sinal de voz, sinal de rádio, entre outros.

Os sinais estão divididos basicamente em dois grupos:

- Unidimensionais (1D)
- Bidimensionais (2D)

No primeiro caso existe uma variação de energia em função de uma variável, que normalmente é o tempo. No segundo caso, por outro lado, a quantidade de energia é função de 2 variáveis, tais como coordenada horizontal e coordenada vertical. Na prática, sinais 1D contemplam áudio, voz, emissão de rádio, emissões biomédicas e neuronais, entre outros. Já sinais 2D correspondem imagens.

Importante também é definir o termo **processamento**, que constitui qualquer tipo de manipulação, análise, modificação e até mesmo síntese, isto é, fabricação de algo. Desse modo, o processamento de sinal consiste em analisá-lo, modificá-lo, etc.

Existe basicamente duas formas para realizar processamento de sinais:

- Processamento Analógico
- Processamento Digital

Na primeira modalidade são empregados circuitos eletrônicos que utilizam resistores, capacitores, transistores, indutores, entre outros, **mas não há processador**.

Na segunda modalidade, circuitos eletrônicos também são utilizados, mas **o componente base é o processador**.

Isso faz com que ambas as modalidades possuam características distintas. No primeiro caso, isto é, analógico, todo sinal é processado continuamente, sem perdas. Enquanto no segundo caso, isto é, digital, somente pontos específicos são processados, pois o processador trabalha somente com valores pontuais, e não tem capacidade para processar infinitos valores.

Lembramos ainda que circuitos eletrônicos analógicos, são construídos com finalidade específica, como por ex. amplificar a voz, robotizar a voz, e assim por diante. Qualquer modificação no propósito do processamento implica na construção de um novo circuito eletrônico, demandando tempo e custo. Por outro lado, a modificação de propósito no processamento digital implica simplesmente em alterar o software correspondente.

Em suma, temos a seguinte situação: Processamento Analógico

- Vantagem: Trabalha com todo o sinal.
- Desvantagem: Mudança de objetivo significa construir um novo circuito.

Processamento Digital

- Vantagem: Mudança de propósito = mudança no software
- Desvantagem: Não é possível tratar os infinitos pontos do sinal

Certamente temos uma motivação maior para realizar processamento digital ao invés do analógico. Para isso precisamos definir qual é a quantidade de pontos, isto é, valores, que temos de processar em meio aos infinitos valores existentes em um sinal qualquer. Para isso utilizamos uma teoria de fundamental importância, e que possibilitou o processamento digital de sinais não textuais, que é conhecida como Teorema da Amostragem, ou **Teorema de Nyquist**.

Aula 2 - 27/03/2015

O teorema da amostragem, ou o teorema de Nyquist, é um resultado do qual podemos fazer uso para realizar um processamento digital de sinais, sendo que de acordo com ele, é necessário entregar ao processador, um número de pontos, isto é, amostras, que é pelo menos igual ao dobro da máxima frequência presente no sinal analógico respectivo.

Em se tratando de sinais de áudio, por exemplo, que constituem os sinais multimídia mais elementares, existe outro resultado experimental garantindo que o ouvido humano não é capaz de perceber frequências acima de 22050 Hz. Desse modo, com base em ambos os resultados é possível concluir que a taxa de amostragem de 22050 x 2 = 44100 am/seg é suficiente para manipular de modo digital os sinais de áudio, lembrando ainda que as amostras devem ser igualmente espaçadas. O processo de amostragem, faz com que o sinal originalmente contínuo seja caracterizado como sinal discreto.

A questão da discretização realizada no eixo horizontal também é necessária no eixo vertical. Este procedimento, que é chamado **quantização** implica em dividir o eixo das amplitudes (vertical) em uma quantidade finita de pontos e utilizar apenas um deles para cada amostra capturada. Assim, por exemplo, se temos amplitudes, no modo contínuo, variando de -5 até 5 e optamos por uma quantização de 1 em 1, então o valor da amostra que no mundo real é "2.74512" será armazenado na versão discreta como o número 3, pois é o valor de amplitude mais próximo existente no mundo digital.

A quantização, diferentemente da amostragem que é regulada genericamente para qualquer sinal, não é expressa para sinais quaisquer de um modo único. Ao contrário, cada tipo de sinal possui uma quantização ideal. De qualquer forma dizemos que a quantização é de 'X' bits, sendo 'X' obtido experimentalmente de acordo com o tipo de sinal. Quantizações tradicionais são, por exemplo, 8 bits para voz, 16 bits para áudio e música em geral, 24 bits para sinais biomédicos, o que equivale respectivamente a 2^8 = 256, 2^16 = 65.536, 2^24 = 16.777.216

Normalmente, a não ser em casos específicos, os valores são os inteiros correspondentes, admitindo-se que um dos bits seja utilizado para representar o sinal positivo ou negativo. Desse modo, utilizando 8 bits temos valores inteiros que estão na faixa `-128 ~ 127`, utilizando 16 bits `-32.768 ~ 32.767`, e utilizando 24 bits `-8.388.608 ~ 8.388.607`.
Assim sendo, um sinal musical armazenado no computador, por exemplo, é na verdade um vetor de 44.100 pontos a cada segundo, para o qual cada posição do vetor é um valor inteiro na faixa de `-32.768 ~ 32.767`.

Quando convertemos o sinal contínuo em sinal discreto, com base no teorema da amostragem, dizemos que o sinal foi **discretizado**. Por outro lado, quando aplicamos os arredondamentos necessários no eixo das amplitudes, dizemos que o sinal foi **quantizado**. Após a aplicação de ambos os processos, dizemos que o sinal, que era originalmente analógico, passou a ser **digital**.

Demonstração: Existe um fenômeno natural que, embora não constitua uma demonstração teórica é suficiente para exemplificar na prática o teorema da amostragem, trata-se do fenômeno que nos dá a falsa impressão de que as rodas de um veículo em movimento estão girando em sentido contrário ao do veículo. Tendo em vista que o olho humano não é um sistema contínuo de detecção de movimento, assume-se uma situação na qual ocorre amostragem, isto é, somos somente capazes de ver e de codificar um número finito de quadros ao longo do tempo, normalmente cerca de 15 ou 20 imagens por segundo. Assim sendo, quando as rodas do veículo giram em uma taxa baixa as amostras capturadas pelo olho na sequência bastam para perceber o movimento no sentido correto. Por outro lado, quando a velocidade é tal que entre cada amostra as rodas giram mais do que meia volta passamos a ter a percepção errada, que volta a ficar correta caso, entre cada amostra elas girem pouco mais de uma volta e assim por diante. Notamos ainda o caso no qual a taxa de amostragem no olho coincide com exatamente 'N' voltas da roda, sendo N = 1,2, 3, ..., ∞; que implica em perceber erroneamente a roda parada. A solução que conduz a percepção correta requer que entre cada ciclo seja possível amostrar pelo menos mais uma vez, o que implica que a taxa de amostragem, para percepção correta, deve ser pelo menos o dobro do número de ciclos, isto é, da máxima frequência.

A percepção errônea corresponde a uma frequência fantasma que, formalmente recebe o nome de **Aliasing**. É muito importante notar que o circuito eletrônico responsável por amostrar e quantizar sinais do meio externo deve ter uma precisão extrema. Isso significa que no momento exato de capturar cada amostra o circuito não pode falhar, ou demorar, pois caso contrário outro valor de amplitude será capturado, confundindo o momento ideal com outro produzindo a distorção do sinal, possível aliasing entre outros efeitos danosos que inviabilizam por completo o processamento digital do sinal. Desse modo, é necessário que exista um dispositivo, que normalmente é um **processador dedicado**, para realizar a amostragem. Este processador está, no caso de um sinal de áudio por exemplo, na placa de áudio. No caso de uma câmera digital este dispositivo está na respectiva placa de captura de imagens, e assim por diante. De qualquer forma não devemos tentar utilizar o processador principal da máquina para controlar a amostragem, pois certamente o sistema operacional suspenderia o processo que cuida da amostragem para atender, de tempos em tempos, outros processos, tais como os que cuidam da rede, de entrada e saída no teclado e mouse, entre outros.

Notamos que independentemente do processador que controla a amostragem, ele só pode trabalhar com os dados já digitalizados, e portanto a amostragem e a quantização necessitam adicionalmente ao processador um circuito eletrônico conhecido como **Conversor Analógico-Digital (AD)**. No caso de desejarmos

algum processamento ou modificação dos dados digitalizados, o respectivo software será executado no processador principal da máquina de modo que a comunicação entre os componentes eletrônicos envolvidos é dada por um conjuntos de vias de comunicação elétrica chamada de **barramento**, conforme estudaremos a diante.

MINI PROVA 1 (PRÓXIMA AULA)

- ELABORAR UM TEXTO COM 2/3 PARÁGRAFOS CURTOS DISCUTINDO AS IMPLICAÇÕES, AS VANTAGENS E AS DESVANTAGENS DE REALIZAR PROCESSAMENTO DIGITAL DE SINAIS FRENTE A ABORDAGEM ANALÓGICA.

Aula 3 - 10/04/2015

Questões relativas a propagação das ondas

Fundamentos de comunicação de dados Em arquitetura de computadores estudaremos as interligações e as diversas disposições dos componentes eletrônicos que formam um computador. Esses componentes estão interligados e portanto se comunicam por meio de sinais elétricos, mesmo que a distância entre eles seja mínima. Esta comunicação é também alvo do nosso estudo pois normalmente impõe barreiras no que se refere a distância e aos locais de conexão dos componentes diversos. Os mesmos conceitos aqui discutidos servem de base para o entendimento da comunicação entre computadores e outros aparelhos.
Inicialmente é importante lembrar que qualquer condutor de eletricidade, tal como cobre, a prata, o ouro, entre outros, possui o seu comportamento modelado em função de 3 elementos, isto é, 3 grandezas físicas:

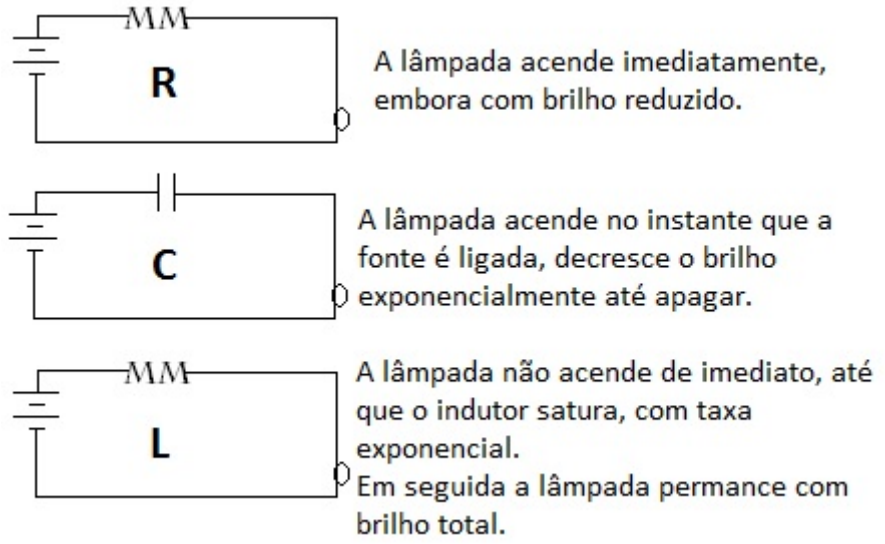
- Resistência: `R`
- Capacitância: `C`
- Indutância: `L`

A resistência, também chamada de resistência ôhmica interfere de modo a fazer uma força contrária a da passagem da corrente elétrica, independentemente do sinal que atravessa o condutor, ser contínuo, alternado, com alta ou baixa amplitude, entre outros fatores. Desse modo, a atenuação causada pela resistência

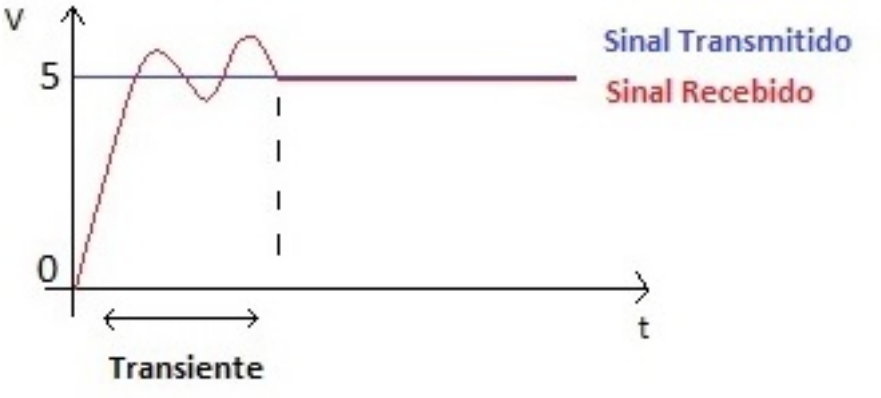
ôhmica é a mesma para qualquer situação, fazendo com que ela seja uma característica não reativa. Usualmente representamos a resistência ôhmica com o símbolo `R`.

Com relação a capacitância, normalmente simbolizada com a letra `C`, temos um efeito distinto da resistência. Em corrente contínua um capacitor, que é o componente eletrônico responsável por prover capacitância, permite a passagem de corrente elétrica até que ele atinja a sua carga completa que é função do seu valor dado em farads.

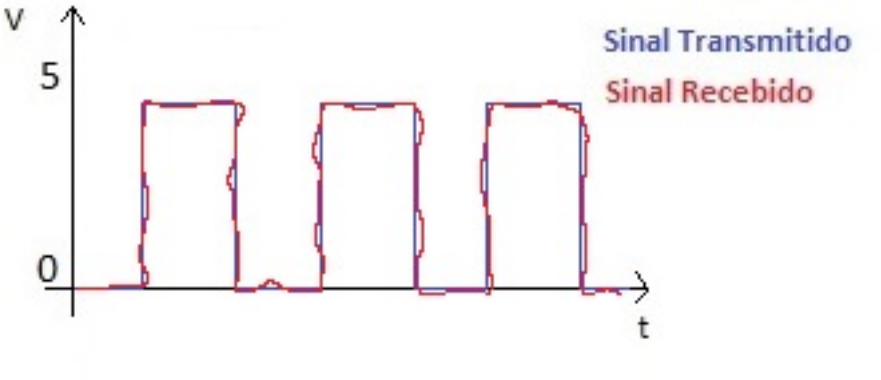
O indultor, por outro lado, apresenta um comportamento inverso ao do capacitor. Em um circuito elétrico de corrente contínua, o indultor oferece uma oposição a passagem da corrente quando ela é aplicada, até o ponto em que, após alguns instantes, dependendo do valor do indultor medido em Henrys, ele se satura, permitindo assim o fluxo integral da corrente elétrica.



Em vista dos comentários acima, e entendendo qualquer condutor possui as propriedades `R`, `L` e `C` intrínsecas, é fácil entender que quando transmitimos, por meio de um condutor, um sinal elétrico de um ponto ao outro, estamos sujeitos aos 3 elementos físicos retromencionados. Conforme podemos observar na figura abaixo o sinal contínuo transmitido é diferente do sinal recebido, enquanto a situação de `C` e de `L` não atinja estabilidade. Este intervalo de tempo é conhecido como **transiente**.



Tendo em vista o comportamento do condutor em corrente contínua, conforme o gráfico acima, que é conhecido como gráfico de resposta ao degrau, podemos facilmente compreender o comportamento do condutor em corrente alternada, conforme o gráfico a seguir.



Constatamos, pelo exame do gráfico acima, que a corrente alternada transmitida, representada na forma de uma onda retangular, é simplesmente um conjunto de degraus, isto é, transições entre 0 e 5 volts. Assim sendo, o sinal recebido, possui inúmeros períodos transientes em torno de cada degrau. Ocasionando a curva do sinal recebido no gráfico. Concluindo, podemos entender que a rigor, não existe transmissão efetiva de uma onda quadrada, pois ela contém sempre inúmeros transientes que correspondem a um conjunto de oscilações senoidais. O estudo mais aprofundado do tema mostra que uma onda quadrada ou retangular, assim como qualquer outra forma de onda não senoidal, é uma soma de infinitas senóides, cada uma com sua frequência e amplitude específica. Este estudo, tem os seus resultados assegurados por meio das **séries de Fourier**. Importante ainda é compreender que com base nas séries de Fourier, o fato do sinal recebido ser diferente do transmitido, ocorre porque as senóides de frequências mais altas não conseguem alcançar o ponto receptor do sinal devido as propriedades `L` e `C` do condutor. Em geral, cada condutor com seus tamanhos

e demais particularidades é equivalente a um conjunto `R`, `L` e `C` que, na prática, não oferece alteração substancial no sinal transmitido desde que cada degrau não tenha um período de duração muito pequeno, isto é, uma frequência muito alta. A frequência limite das senóides que atravessam o condutor sem alteração substancial recebe o nome de **largura de banda**.

Dentre as inúmeras possiblidades de efetuar transmissão de dados entre dois pontos, a opção que utiliza a base 2 é a melhor, pois próximos aos degraus existentes no sinal, as distorções oferecem uma margem pequena de erro, pois assumindo por exemplo que os níveis lógicos 0 e 1 correspondam na realidade física a 0 e a 5 volts respectivamente basta um único limiar em 2,5 volts por exemplo. Níveis de tensão abaixo do limiar são interpretados como lógica 0, e níveis acima do limiar como lógica 1. Se por outro lado tivéssemos 3 ou mais níveis precisaríamos de 2 ou mais limiares respectivamente, aumentando assim a possibilidade de erro interpretação. O caso extremo é o que possui **infinitos níveis**, correspondendo assim ao **sinal analógico**. Além disso, a base 2 oferece possiblidades mais simples para realizar cálculos diversos nos processadores e desse modo é a opção adotada, para qual não existe qualquer previsão ou intenção de alteração.

A possiblidade de uso da base 2 foi se consolidando ao longo das diversas gerações de computadores ao longo dos tempos. Particularmente, todas as máquinas e equipamentos de desenvolvidos com a finalidade de realizar cálculos até o ano de 1945 englobam o que chamamos de **Geração 0** de computadores, na qual incluímos basicamente as réguas de cálculo e as máquinas calculadoras, que atualmente constituem peças de museu. Entre 1945 e 1955, passamos a ter a **Geração 1** de computadores, que eram basicamente constituídos de válvulas eletrônicas. Tais computadores normalmente ocupavam espaços físicos imensos e nos quais a temperatura média chegava aos 50~60 °C. Tal fato, implicava em manter os referidos computadores em gigantescos armários térmicos, sendo ainda necessário realizar trocas de cabeamento e conexões para que o computador fosse reprogramado.

Com os avanços nas pesquisas, alcançamos a **Geração 2** de computadores que vão de 1955 a 1965. Nesta geração a base dos circuitos foi o transistor semicondutor em sua versão discreta. Os transistores constituem componentes eletrônicos com função básica de amplificação e comutação de sinais, sendo constituídos basicamente de 1 dentre 2 minerais naturais: **Silício** e **Germânio**, ambos constantes da tabela periódica. Para o correto funcionamento, os transistores são constituídos de 1 desses minerais especialmente preparados com base num processo químico conhecido como dopagem e que tem a função de acrescentar impurezas aos minerais necessárias ao funcionamento do componente. O processo, que se tornou de baixo custo, requer maquinário considerado como muito específico, tendo sido inicialmente desenvolvido no Bell Labs nos EUA. Uma característica peculiar dos transistores é que eles requerem tensões e correntes baixíssimas para funcionar, entretanto, são fisicamente quase microscópicos. Visando, desse modo, permitir o uso industrial e comercial deles, faz-se necessário encapsulá-los

em uma massa plástica que os torna muito maiores. Assim o transistor discreto, como é popularmente chamado, é muito maior que o transistor real.

Com o advento da tecnologia mecânica, a indústria passou a ser capaz de manipular mais facilmente elementos quase microscópicos, dessa forma foi possível passar a conectar vários desses elementos e assim encapsulá-los em um único invólucro. Desse modo, surgiu então entre 1965 e 1980 a **Geração 3** de computadores que era basicamente constituída de conjuntos de transistores encapsulados juntos, fato este que deu origem ao nome **Circuito Integrado**. Após a década de 80, o avanço da tecnologia permitiu a construção de máquinas capazes de agrupar milhões de transistores em um único circuito integrado. Esse fato conduziu a **Geração 4** de computadores, também conhecida como geração de Very Large Scale Integration (VLSI). Os avanços tecnológicos, tanto em nível de fabricação de componente quanto em nível de programação deles via software, fez com que após o ano 2000 uma nova geração, conhecida como **Geração 5** de computadores fosse definida. Nela, encontramos muitos milhões de transistores e outros componentes em um único circuito integrado que conta ainda com reduções drásticas de tamanho físico. Essa geração é também conhecida como das máquinas de baixa potência, ou máquinas "invisíveis" ou até **máquinas descartáveis**. Exemplos são os smartphones, tablets, entre outros, inclusive cartões de Radio Frequency Identification(RFID).

A atuação conjunta software-hardware, somada ao desenvolvimento técnico-mecânico de fabricação dos circuitos integrados fez com que as máquinas de baixa potência ou descartáveis estivessem disseminadas em toda a indústria eletrônica, passando então a compor os mais diversos aparelhos, tais como, geladeiras, máquinas de lavar, brinquedos, relógios, entre outros. Desenvolveu-se inclusive uma linha de pesquisa para tratar particularmente desse tema de pesquisa que é conhecida como computação **ubíqua** ou **pervasiva**.

MINI PROVA 2 (PRÓXIMA AULA)

- ELABORE UM PEQUENO TEXTO RESUMINDO E DISCUTINDO OS CONCEITOS DE CADA UMA DAS 5 GERAÇÕES DE COMPUTADORES.

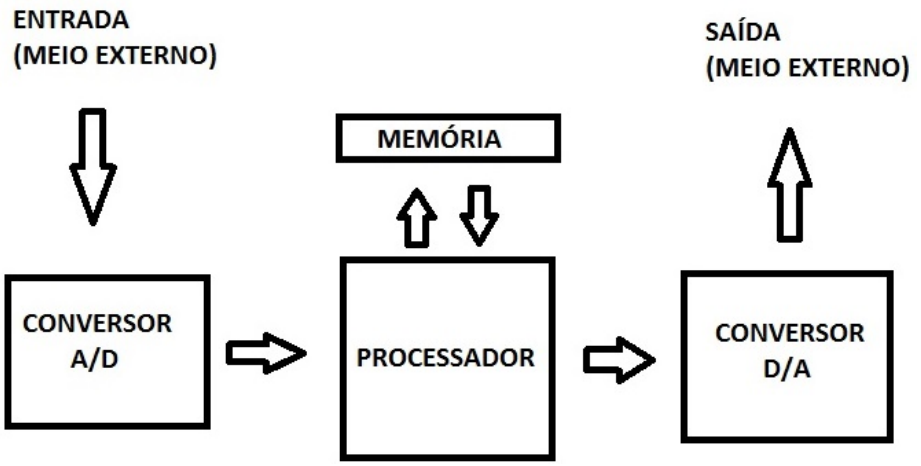
Aula 4

Tecnologias Emergentes de 5ª Geração

Conforme vimos anteriormente, a 5ª Geração de computadores engloba os dispositivos que possuem tecnologia de processamento digital com baixo consumo

de energia e tamanho extremamente reduzido. Particularmente, podemos inserir em tal contexto alguns dispositivos para processamento dedicado, sendo que entre eles temos os **Digital Signal Processors (DSPs)** e também **Field Programmable Gate Array (FPGAs)**. Estes últimos podem ainda, conforme a finalidade especifica ter a denominação de **Control Programmable Logic Devices (CPLDs)**.

Os DSPs são **processadores**, normalmente disponibilizados em placas eletrônicas que trazem outros circuitos de apoio, dedicados para realizar operações de modo veloz e para uso em **tempo real**, isto é, provendo **atraso fixo** entre a entrada e a saída. Os principais fabricantes de DSPs são **Texas Instruments** e a **Motorola**. O diagrama de blocos de DSPs é, de modo simplificado, o seguinte:



Exemplos do uso DSPs são: Modificações realizadas de modo digital, em áudio para uso on the fly, sistemas de controle e correção de trajetória em freios ABS, entre outro. Com relação as FPGAs ou CPLDs, fabricadas, principalmente, pela **Altera** e/ou **XILINX**, temos um funcionamento bastante particular e que a rigor é extremante versátil. Na realidade, esses dispositivos não constituem processadores. Ao contrário, podem ser configurados para atuar como processadores e também como outros tipos de circuitos digitais, o principal motivo para utilizá-los é que eles possibilitam uma otimização pois o software utilizado para programá-los faz na verdade ligações internas ao componente em nível de hardware. Isso implica em termos um circuito que pode ser literalmente montado via software.

Em vista da versatilidade, normalmente as FPGAs possuem custo mais elevado do que os DSPs. Além disso, tem sido comum no mercado FPGAs vendidas em conjunto com outros dispositivos, tais como telas de cristal líquido, touchscreen, miniteclados, entre outros. Somente as FPGAs permitem total e completa proteção à propriedade intelectual dos desenvolvedores de software, pois é possível queimar, isto é, interromper definitivamente o caminho de dados de comunicação com o meio externo. Dessa forma, se for de interesse podemos isolar o

projeto desenvolvido definitivamente.

Artigo 1

- Elaborar um artigo cientifico de 3 ou 4 páginas em inglês discutindo vantagens, desvantagens e aplicações das FPGAs.
- O Artigo deve seguir a seguinte formatação.
- TITULO
- NOME, EMAIL, ENDEREÇO PARA CONTATO
- ABSTRACT (3,4 LINHAS)
- INTRODUCTION COM 2/3 PARÁGRAFOS
- DISCUSSION 2/3 PARÁGRAFOS
- CONCLUSSIONS 1 PARÁGRAFO
- REFERENCES.
- Sites confiáveis para pesquisas e artigos científicos.
- www.ieee.org -> IEEEXplorer
- www.sciencedirect.com
- isiknowledge.com
- Editor para fazer o artigo **LaTeX**.

Representação de dados nos computadores

Conforme já discutimos anteriormente, **qualquer dado** constitui um conjunto de **números**, que pode descrever texto, sinais multimídia 1D e 2D. Particularmente, no caso de **texto**, a representação computacional é realizada por meio de **números inteiros**. Por outro lado, **sinais multimídias** podem utilizar **inteiros** ou **reais**, também chamados de **pontos flutuantes**.

Representação de Texto:

A representação de texto em computadores digitais, baseada em valores **inteiros**, é regulada pelo **American Standard Code for Information Interchange (ASCII)**. O referido código, constitui um padrão internacional para simbolizar caracteres diversos que compõe um texto. Procurando na tabela ASCII, podemos conhecer o código designado para cada símbolo.
Como exemplo temos os valores: 65, 66, 67, 97, 98 representando respectivamente `A`, `B`, `C`, `a`, `b`, de qualquer forma, **todos os caracteres do código ASCII são números que quando convertidos para a base 2, utilizam 8 bits**, isto é, são **valores inteiros na faixa de 0 a 255**. Mesmo que os caracteres de um texto possam eventualmente ser escritos na base 2 com menos do que 8 bits significativos, o padrão determina o uso dos 8, sendo q nesse caso os primeiros bits são 0.

Exemplo:

- representar a string "Eu amo chocolate" para uso em um computador digital. Indique os valores na base 10.
  69-85-32-65-77-79-32-67-72-79-67-79-76-65-84-69

Devemos notar que todas as linguagens de programação nos trazem a possibliidade de acessar **1 byte** de memória por meio do uso de variáveis do tipo `char`. Normalmente uma variável `char` portanto, ocupa **1 byte independentemente de características de hardware, tal como a arquitetura**.

OBS: As diversas sequências de 0 e 1 podem ser comprimidas, isto é, podem utilizar um número menor de bits, em um processo sem perdas, reversível ou loss less, planejando convenientemente dicionários de dados com base em estatísticas de ocorrências de 0 e 1. Um dos procedimentos mais comuns para comprimir uma string é conhecido como **Código de Huffman**.

OBS 2: Lembramos que diversas linguagens de programação permitem o acesso a bits específicos de uma determinado byte. Para isso utilizamos os operadores lógicos E bit-a-bit e OU bit-a-bit que em linguagem C são representados respectivamente por `&` e `|`.

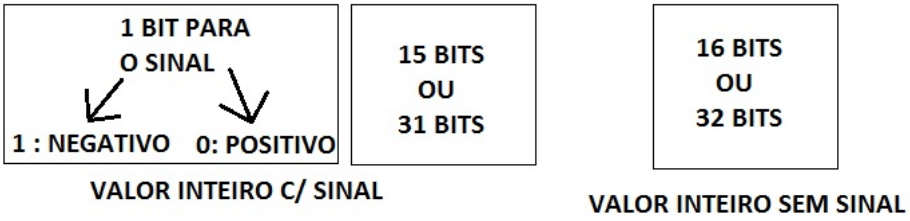
Ex:

```
...
x = 65;
x = x&10;
...
// Após executar o código acima o resultado será:
65 -> 01000001
10 -> 00001010
--------------
 X -> 00000000
```

- Como faríamos para, com base nos operadores acima, isolar, por exemplo, o bit menos significativo de `x`?
  Basta fazer um E bit-a-bit com o 00000100 (4), pois as posições com 0 anularam as posições respectivas da variável `x`, restando somente o bit de interesse. Esse bit restante pode ainda ser rotacionado para esquerda ou para a direita com os operadores `<<` e `>>`.

Representação de dados oriundos de sinais 1D ou 2D por inteiros:

Considerável parte dos sinais manipulados nos computadores utilizam a representação por valores inteiros de modo a corresponder com a quantização adotada. Assim, se um sinal é quantizado em amplitude com `n` bits, possui representação na qual 2^n possibilidades existentes, conforme já discutimos anteriormente. De modo geral, **uma variável declarada como inteira, possui 2 ou 4 bytes, dependendo da linguagem e da arquitetura**. De qualquer forma, se não há menção especifica na declaração, o primeiro bit é usado para representar o sinal e os demais para representa o número em módulo. Assim por exemplo temos `1111111111111111` representa o número `65.535` caso a variável que o carrega tenha sido declarada como um `unsigned int`. Caso a variável seja declarada somente como `int`, então ela possui sinal. Desse modo o seu valor em módulo não leva em conta o primeiro bit e no caso do exemplo acima, ele será `-32768`. Por padrão, quando a variável possui sinal e o primeiro bit é 1, ela é negativa, sendo positiva se o primeiro bit for 0. Estabelece-se desse modo a regra:



- O modelo acima para representação de inteiros é conhecido como modelo **sinal-magnitude**.

MINI PROVA 3 (PRÓXIMA AULA)

- CONSTRUIR UM PROGRAMA EM C PARA SOLICITAR E LER NO TECLADO UM VALOR DE 8 BITS EM UMA ÚNICA VARIÁVEL E APRESENTAR UMA MENSAGEM NA TELA INFORMANDO SE O TERCEIRO BIT MENOS SIGNIFICATIVO É 0 OU 1.

Aula 5

Codificação e representação de conteúdo multimídia com valores inteiros

Do mesmo modo como trabalhamos anteriormente na codificação de texto, trabalharemos no padrão **sinal-magnitude** visando codificar sinais diversos, que possuem correspondentes analógicos, utilizando **amostragem** e **quantização**. Lembramos ainda que, embora esta definição não seja precisa, na prática podemos encarar um texto como sendo um sinal originalmente discreto e para o qual não existe correspondente analógico.

Procedemos a seguir com a codificação de um sinal 1D. Exemplo: Codificar, utilizando valores inteiros no padrão sinal-magnitude, com taxa de amostragem de 20 mil amostras por segundo e com quantização de 16bits, o seguinte sinal acústico. Codifique, para facilitar, as 10 primeiras amostras do sinal.

- **Solução**:

Usaremos a função: f(t)=30000.sen(2pi.4000t). Onde 30000 é amplitude e não interfere no período.

O sinal em questão possui o seguinte aspecto:

`**TODO** gráfico senoide de -30k a 30k`

> Observamos que o sinal em questão pode ser quantizado com 16bits pois possui valores variando de -30000 até 30000 e com 16bits podemos representar inteiros com sinal na faixa de -32768 até 32767.

É importante também verificarmos se a taxa de **amostragem** proposta condiz com o sinal analógico lembramos que, de acordo com o **Teorema da Amostragem**, precisamos no mínimo de uma quantidade de pontos por segundo igual ao **dobro** da máxima frequência presente no sinal, sendo os pontos igualmente espaçados. Assim, considerando que `p` é o período do sinal acima, constatamos o que segue:

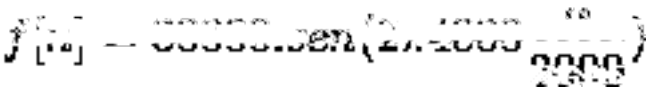
```
f(t+p) = f(t), logo

sen(2pi4000(t+p)=sen(2pi4000t)
2pi4000(t+p) = 2pi4000t + 2kpi
Considerando K = 1.
2pi4000t + 2pi4000p = 2pi4000t + 2pi
2pi4000p = 2pi
4000p = 1
p = 1/4000 => F=1/p = 4000Hz ou 4Khz
```

Constatamos assim que a frequência do sinal é de 4000 Hertz e portanto a taxa de amostragem mínima é de 8000 amostras por segundo. Tendo em vista que solicitamos no enunciado 20000 amostras por segundo não teremos problemas.

> **OBS**: Toda vez que um sinal analógico estiver no formato `f(t) = A.sen(2pi.F.t)`, então `F` é sua frequência.

Para amostrarmos o sinal analógico conforme solicitado, fracionamos o tempo de 1 segundo em 20000 amostras, conforme o enunciado. Utilizando ainda, a variável `n` no lugar de `t` e colchetes no lugar de parênteses, visando manter a padronização indicativa de sinal digital, temos:

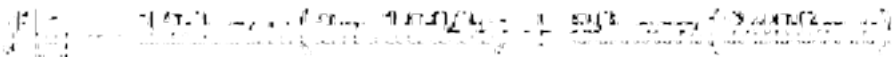


```
f[1] = 28531,695
f[2] = 17633,557
f[3] = -17633,557
f[4] = -28531,695
f[5] = 0
f[6] = 28531,695
f[7] = 17633,557
f[8] = -17633,557
f[9] = -28531,695
f[10] = 0
```

- Lembrando que os valores devem ser representados como inteiros então a real representação é a seguinte:

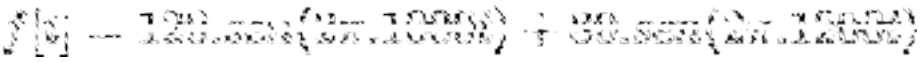
```
f[1] = 28532
f[2] = 17634
f[3] = -17634
f[4] = -28532
f[5] = 0
f[6] = 28532
f[7] = 17634
f[8] = -17634
f[9] = -28532
f[10] = 0
```

Exemplo2: Idem anterior, quantizando o sinal com 8 bits no padrão sinal-magnitude e utilizando a menor taxa de amostragem possível para que não ocorra aliasing.



- Solução:

Começaremos "melhorando" a equação:



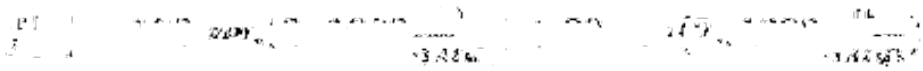
Após escrevermos o sinal no formato padrão (acima) vamos examinar a **quantização**.

Verificamos que caso ambas as senóides atinjam ao mesmo tempo os seus valores extremos, podemos ter resultados variando -200 até 200 usando 8 bits, somente

podemos representar valores de -128 à 127 de forma que a quantização proposta é **insuficiente**. Se forçamos a quantização com 8 bits teremos trecho do sinal que apresentarão **saturação**, isto é, ficarão limitados ao valor máximo possível de ser representado, conforme exemplificado a seguir.

TODO grafico "saturado"

Aumentando a quantização para 9bits percebemos que o problema estará resolvido pois valores na faixa de -256 até 255 serão possíveis. Em computadores corriqueiros(tradicionais), o padrão é a existência de variáveis que implicam em representações numéricas com quantidade de bits múltipla de 8. Assim sendo, precisaríamos utilizar uma quantização de 16bits. Por outro lado, em placas dedicadas de processamento, tais como os DSPs, as FPGAs e as CPLDs, é possível planejar a quantização e acessar a memória com o número de bits variados, sendo comum por exemplo o uso de 4bits, 10bits, 12bits, 24bits e até 40bits. Assim sendo a opção pela quantização depende do conjunto Software-Hardware disponível. Prosseguiremos com o exemplo assumindo o uso de uma placa DSP que disponibiliza a quantização com 10bits, pela qual podemos ter valores inteiros na faixa de -512 até 511. Uma vez solucionado o problema da quantização vamos analisar o espectro do sinal, isto é, o seu conteúdo de frequências para que seja possível definir a taxa de amostragem.
Os dois subsinais que compõe o sinal considerado possuem frequência de 1000Hz e 1200Hz. Assim, sendo 1200 a maior frequência, a taxa de amostragem mínima é de 1200x2 que é igual a 2400 amostras por segundo. Portanto a versão discreta do sinal é:



- As dez primeiras amostras são:

```
f[1] = 60
f[2] = -103,923
f[3] = 120
f[4] = -103,923
f[5] = 60
f[6] = 0
f[7] = -60
f[8] = 103,923
f[9] = -120
f[10] = 103,923
```

- Procedendo com a quantização os resultados são os seguintes:

```
f[1] = 60
f[2] = -104
f[3] = 120
f[4] = -104
f[5] = 60
f[6] = 0
f[7] = -60
f[8] = 104
f[9] = -120
f[10] = 104
```

OBS: Os exemplos acima constituíram modelos bem definidos, para os quais é possivel analisar e determinar a taxa de amostragem e a quantização necessárias. Por outro lado, para sinais oriundos do meio externo e que não possuam modelo bem definido, devemos analisar ao menos as suas essências ou tipos. Por exemplo, se o sinal é de áudio sabemos que 44100 amostras por segundo com 16bits de quantização basta. Para voz, 8000 amostras por segundo com 8 bits de quantização são o suficiente.

Representação computacional de valores reais

Visando registrar valores reais, também conhecidos como de ponto flutuante, oriundos tanto da digitalização de sinais do meio externo quanto de variáveis e cálculos existentes em Softwares diversos, podemos utilizar as representações adequadas para tais casos. Sabe-se que números reais podem ser representados genericamente na forma [illegible], sendo `M` a mantissa, isto é, a parte fracionária, `B` a base e `e` o expoente. Particularmente no caso dos computadores o padrão internacional IEEE754 implica que a representação de valores reais é realizada primeiramente normalizando a mantissa para que o expoente seja positivo e usando a base 2. Sendo assim, o modelo representacional fica sendo [illegible]. Como exemplo, vamos representar o número 5.75 utilizando o referido padrão. Devemos, obviamente, converter o valor em questão para a base 2.
Para realizar a conversão devemos trabalhar separadamente com as partes **inteira** e **fracionária**. A conversão da parte inteira implica em realizar divisões sucessivas por 2 até que o cociente se torne 0 e recuperar o resto inteiro das divisões em ordem contrária. Para a parte fracionaria, devemos multiplica-las sucessivamente por 2, usando a parte fracionaria da multiplicação anterior para realizar a próxima multiplicação, repetindo o processo até que a parte fracionária seja 0. Em seguida devemos recuperar as partes inteiras de cada multiplicação na ordem em que elas ocorreram. Finalmente concatenamos a parte inteira na

base 2 com a parte fracionaria na base 2, usando o ponto decimal entre elas. O procedimento está resumido no diagrama a seguir:

TODO diagrama conversão de real para base 2

O valor 101.11 é a representação correta, na base 2, do valor 5.75 entretanto para representa-lo no computador precisamos adequá-lo ao padrão IEEE. Para isso é necessário, conforme observamos acima, escrever valor na forma [illegible]. Para isso aplicamos o processo de normalização que consiste em deslocar as casas decimais em número já na base 2 de modo que antes do ponto decimal tenhamos apenas um digito 0. Assim fazemos 101.11 = 0.10111x2^3 (usamos o 2 porque é a base, por exemplo 3,14 = 0.314x10^1 porque é na base 10).
Após a normalização o número está pronto para ser representado em um computador digital. Para isso, escolhemos um dentro os dois formatos disponíveis na grande maioria dos conjunto Software-Hardware e acessíveis por meio das diversas linguagens de programação:

- Float:
  - 1bit para o sinal: 0 = positivo e 1 = negativo
  - 8bits para o expoente
  - 23bits para a mantissa
- Double
  - 1bit para o sinal: 0 = positivo e 1 = negativo
  - 11bits para o expoente
  - 52bits para a mantissa

Desse modo, variáveis declaradas como `float` ocupam 1+8+23 = 32bits(4bytes) e variáveis declaras como double ocupam 1+11+52 = 64bits(8bytes). No caso do nosso exemplo, isto é, 0.10111x2^3, a sua representação como float é a seguinte:

> 0 para o sinal; expoente 3 então (11), logo 00000011; e 00000000000000000010111 para a mantissa;

No caso de representarmos o nosso exemplo como um double, devemos fazer o mesmo acima, porem lembrando que é 11bits para o expoente e 52bits para mantissa.

Concluímos desse modo que o nosso número em exemplo pode ser representado com folga por meio de um `float`, não requerendo o uso de um `double`. De modo genérico o padrão `IEEE754`, implica que computacionalmente temos a seguinte situação:

TODO: UNIVERSO DOS VALORES REAIS

MINI PROVA 4 (PRÓXIMA AULA)

- REPRESENTAR O VALOR 2.718 NAS PRECISÕES FLOAT E DOUBLE NO PADRÃO IEEE754

Aula 6

TAXONOMIA

Uma vez que completamos os estudos focados na representação de valores `int`, `float`, `double` passamos a dar atenção ao conceito de **taxonomia** que implica na topologia(interligação dos elementos processados e memoria) para posteriormente estudarmos os processadores internamente diversas classificações existemem ambiente academico sendo que nenhum é considerado ideal , de qualquer forma a classificação taxonomica mais utilizada foi a publicada por michael flynn em 1986 ficando conhecida como taxonomia de flynn. Essa taxonomia divide em 4 tipos as possibilidades de ligação entre processadore, com a função basica de realizar calculos e memória com a função basica de armazenar dados e instruções. particularmente temos :

- **single instruction single data (SISD)**:

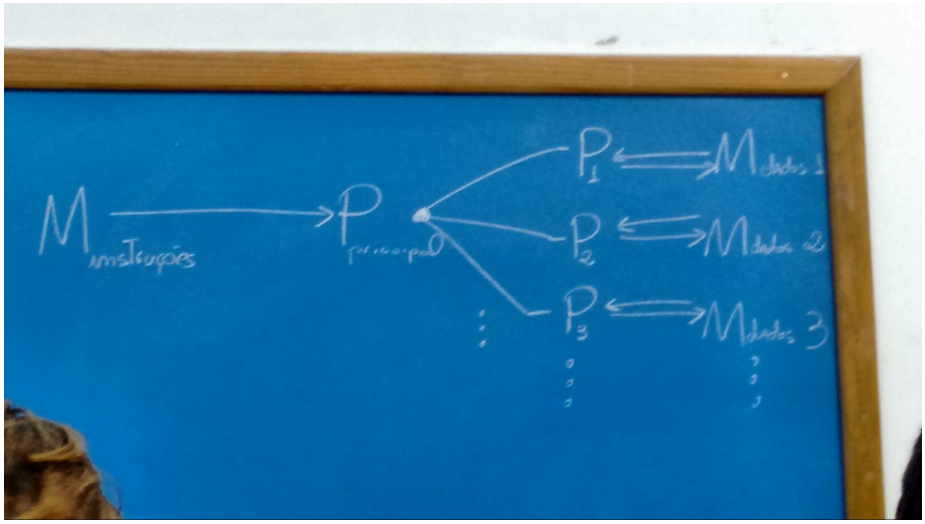
Trata-se do modelo mais simples e utiizado na grande maioria dos computadore pessoais. pode ser representado como segue, sendo P o simbolo para processador e M o simbolo para memória,

P⇄M

A simbologia acima significa que computador e memoria e o processador requisita a proxima instrução a ser executada buscando-a na memoria. em seguida a intrução é enviada da memoria para o processador, o processador requisita ainda os dados, isto é os operandos envolvidos com a instrução a ser executada, esses dados são então enviados da memoria para o processador. O processador requisita ainda os dados, isto é, os operandos envolvidos com a instrução a ser executada. Esses dados são então enviados da memória para o processador, que por fim, executa a instrução possivelmente devolvendo o resultado para a memória. As intruções normalmente são de calculos, de comparações, ou de desvio conforme estudaremos adiante posteriormente as comparações e os calculos são feitos com base em valores numéricos, isto é os dados obtidos na memoria, esssa conversa entre um unico processador e um unico bloco de memoria constitui o modelo mais simples de taxonomia

- **Single Instruction Multipple Data(SIMD)**:

Constitue um modelo um pouco mais aprimorado sendo representado da seguinte forma:

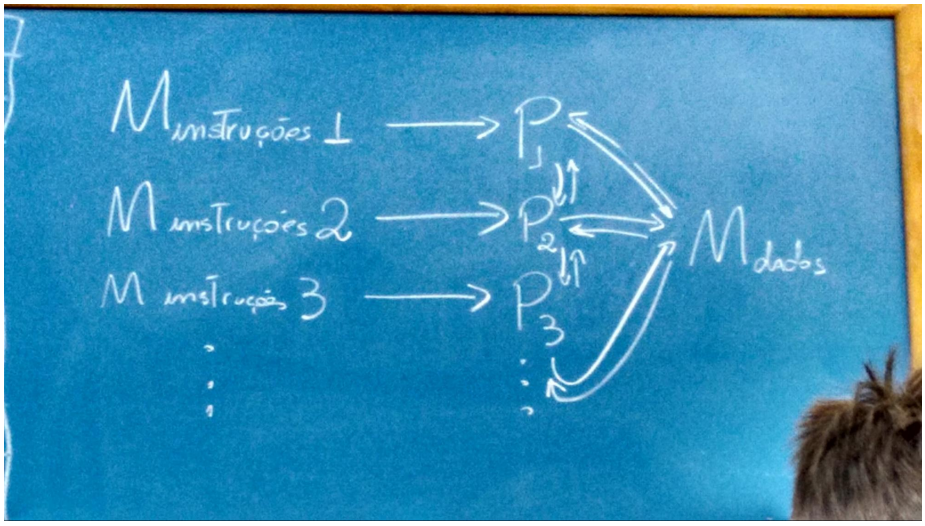


Neste modelo tradicionalmente utilizado nas Graphics Processing Units (GPUs) a porção de memoria disponivel é dividida em duas partes sendo quue uma somente armazena instruções e outra que por sua vez é subdivida em blocos armazena dados.. tem-se ainda um processador dedicado a recolher da memoria de intruções cada instrução a ser executada e distribui-as aos demais processadores a medida que estiverem livres. cada um desses processadores conversa com o seu bloco de memoria de dados, buscando operandos e enviando resultados. caracteriza-se deste modo um principio de paralelismo pois em uma mesma janela de tempo os processadores P1,P2,P3 etc estão cada qual executando determinadas instruções fica, assim clara a aplicabilidade deste modelo no pro-

cessamento gráfico em que imagens precisam ser sintetizadas com consideravel velocidade.

- **Multiple Instruction Single Data (MISD)**:

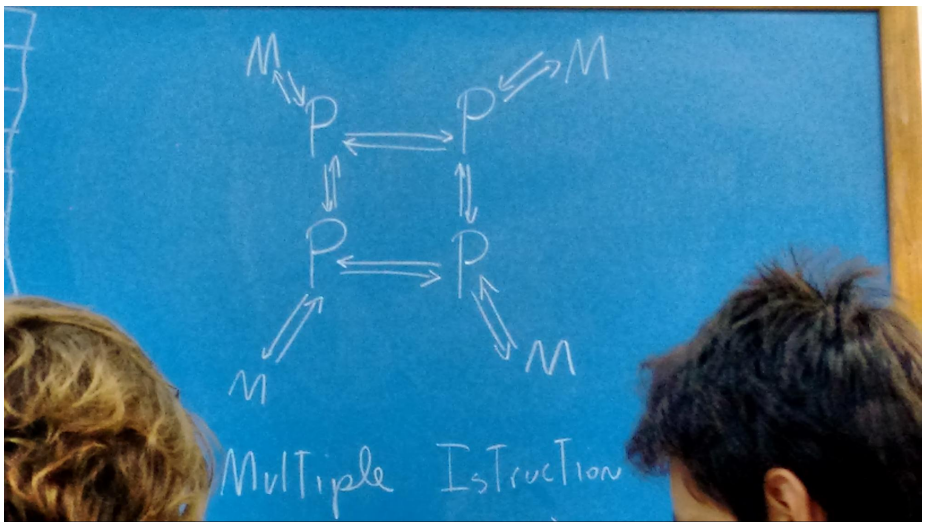
Neste modelo, representado a seguir, temos vários blocos de memoria destinados ao armazenamento de instruções mas somente um bloco destinado ao armazenamento de dados. para cada bloco de memoria de instruções existe um processador correspondente. neste modelo o foco é o processamento divesificadode um mesmo dado em paralelo.



Este modelo é na verdade uma concepção teoria para o qual té o presente momento não existe implementação comercial, havendo somente poucas implementações em nivel de teste industrial e academico.

- **Multiple Instruction Multiple Data (MIMD)**:

Ai na frente coloca o seguinte, trata-se de uma taxonomia na qual os diversos processadores existentes possuem cada qual o seu bloco de memoria que servem para armazenar dados e instruções, alem disso os processadores conversam, entre aspas ne, entre si para a distribução das tarefas. o diagrama a seguir exemplifica esta taxonomia para um caso de 4 elementos de processamentos:



---

os modelos mais simples e mais completos de taxonomias, isto é, SISD e MIMD foram concebidos para finalidadfes genericas de processamento, sendo obviamente versoes nao paralela e massivamente paralelano caso do modelo SIMD, a aplicação mais propicia é aquela para a qual temos a necessidade de processamentos do tipo

```
for(inti=0;i<100;i++)

{

                g[i]=sqrt(x[i]);

}
```

ou seja uma mesma instrução é aplicada em operandos diferentes, por outro lado no modelo MISD a ideia é aplicar diversas instruções em um unico dado ou par de dados, na pratica teriamos então um modelo de taxonomia particularmente util para situações oriundas de códigos tais como:

```
y=sqrt(x);

y2=cos(x);

y3=sin(x);
```

```
y4=exp(x);
```

é possivel observar facilmente o motivo pelo qual o modelo MISD não é implementado comercialmente, ja que beneficiaria que otimizaria sequencias de codigos que estatisticamente são incomuns ou pouco encontrados.

- **MINIPROVA 5**: codifique, isto é apresente o resultado da digitalização tanto em decimal quanto em binário no padrão float IEEE, as 5 primeiras amostras do sinal unidimensional

```
f(t)=120sin(2*pi*175t)
```

utilizando uma taxa de amostragem igual a 3 vezes e meio a máxima frequencia do sinal(175*3,5).

Aula 7

Modelos de Arquiteturas

Do ponto de vista comercial, excluindo modelos teóricos e/ou puramente acadêmicos, podemos considerar a existência de duas arquiteturas de computadores:

- Arquitetura de Von Neumann.
- Arquitetura Harvard.

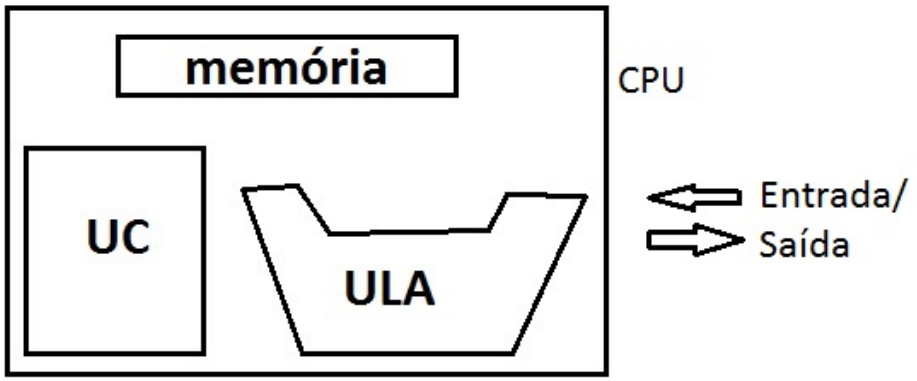
O modelo de **Von Neumann**, assim chamado por ter sido especificado pelo matemático húngaro John Von Neumann, constitui o esquema mais básico de processamento, pelo qual a área de memória é compartilhada para armazenar as instruções a serem executadas e os dados, ou seja, operandos, utilizados em conjunto com tais instruções. Existe, dessa forma, um único barramento, isto é, conjunto de vias elétricas por onde as informações transitam para conduzir dados e instruções. Podemos dizer que não há paralelismo nas tarefas e a sequência de processamento está sujeito a um gargalo, conhecido como gargalo

de Von Neumann, que limita o desenvolvimento das atividades de processamento. Particularmente percebemos que o modelo de Von Neumann caracteriza as máquinas para as quais a taxonomia é do tipo **SISD**.
Em contra partida, o modelo **Harvard** especificado por pesquisadores diversos na universidade Harvard, na década de 40, tem como princípio a existência de regiões de memória separadas para armazenar instruções e dados. Desse modo existe um barramento que interage com uma porção de memória para troca de instruções e outro barramento que interage com uma segunda região de memória visando trocar dados. Entende-se, assim que já existe a ideia de paralelismo pois enquanto transita uma instrução por um barramento, os respectivos dados ou operandos transitam por outro. Nota-se então que este modelo Harvard corresponde a proposta da taxonomia **SIMD**.

Independentemente do modelo de arquitetura, temos sempre a figura da unidade central de processamento (Central Processing Unit – CPU), que contém basicamente duas subunidades:

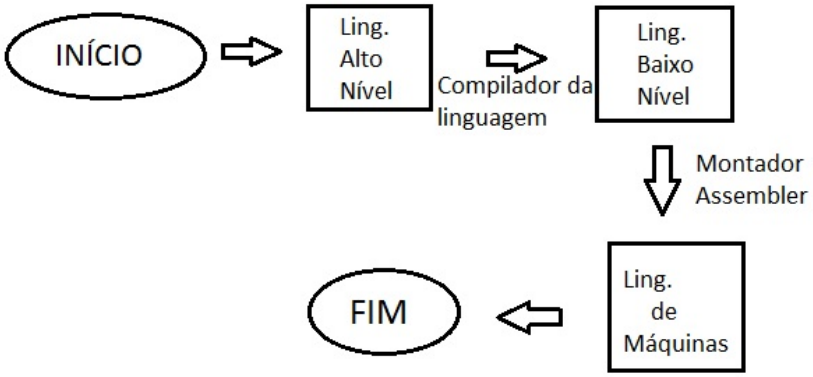
- Unidade de Controle (UC)
- Unidade Lógica e Aritmética (ULA)

A primeira subunidade, isto é **UC**, tem como função comandar a **sequência de processamento**, isto é, qual instrução será executada em qual momento; enquanto que a segunda subunidade, isto é **ULA**, realiza os **cálculos e as comparações lógicas**. Acrescentamos ainda à CPU uma porção de **memória**. Particularmente, pequenos blocos de memória, utilizados dentro do computador costumam possuir velocidade de escrita e leitura **superior** a memória RAM da máquina, sendo chamado de **registradores**. Em geral os registradores armazenam resultados imediatos ou intermediários das operações aritméticas e lógicas.



É de nosso conhecimento, que os processadores somente entendem uma linguagem chamada de **linguagem de máquina**, e que é definida e conhecida

pelos engenheiros eletrônicos e de computação que projetaram o processador. Essa linguagem constitui um conjunto de sinais elétricos digitais que comandam a execução das tarefas no processador. Tendo em vista tratar-se de uma linguagem muito próxima ao hardware que seria inacessível muitas vezes até mesmo para os engenheiros, é sempre definida uma linguagem mais facilmente compreendida que é chamada de **linguagem Assembly**, havendo ainda um conversor, chamado montador, ou **Assembler**, que funciona como um **compilador** traduzindo os comandos Assembly para linguagem de máquina. Cada processador possui seus conjuntos de instruções, sendo que as versões melhorada de uma linha de processadores contém todas as instruções das versões anteriores adicionadas de algumas outras que são otimizadas para determinado fim. Por exemplo, o processador i7 reconhece todas as instruções do i5 e algumas outras. A linguagem Assembly dos processadores, que iremos estudar adiante, não é confortável para programar e desenvolver software no dia-a-dia. Assim sendo, foram criadas as linguagens de programação de alto nível, cada qual com seu compilador que traduz as instruções da linguagem para comandos Assembly. Na figura a seguir, temos um resumo deste procedimento.



Notamos que, cada processador tem o seu conjunto de comandos que são reconhecidos. Dependendo da quantidade e da função especifica do conjunto de comandos de um processador, podemos classificá-lo em um entre dois modelos:

- Reduced Instruction Set Computer - **RISC**
- Complex Instruction Set Computer - **CISC**

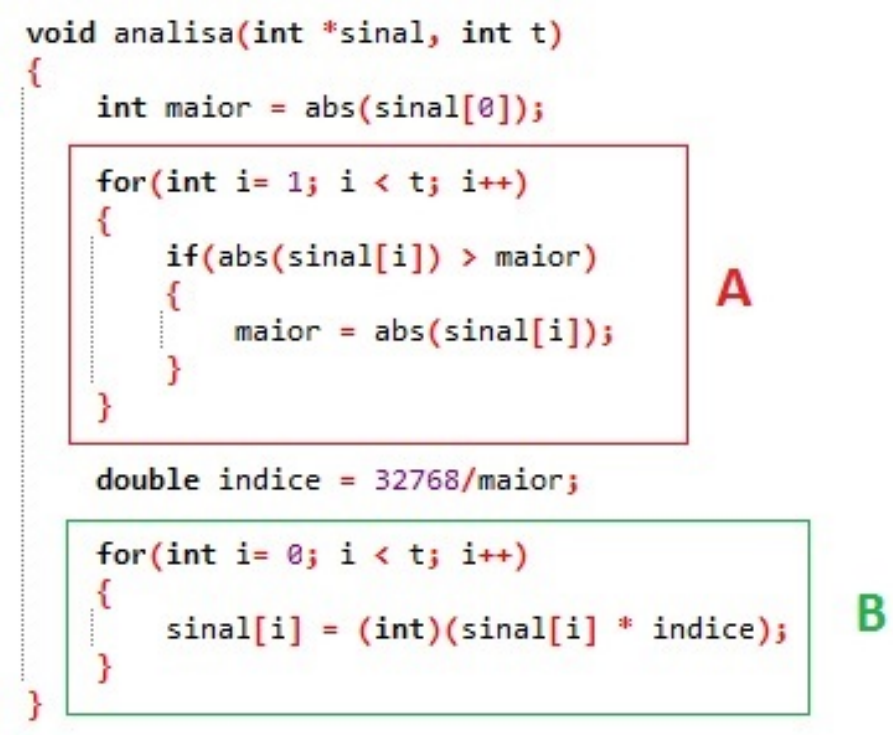
Nas máquinas **RISC**, costumeiramente temos algumas dezenas de comandos para realizar operações mais elementares, sendo que as operações mais específicas são forçosamente implementadas a partir das mais elementares, como exemplo temos as instruções de soma e subtração entre dois operandos. Por outro

lado, as máquinas **CISC** possuem instruções para realizar não somente as operações básicas, mas também as mais complexas ou elaboradas, definidas a partir das básicas, tal como por exemplo uma exponencial. Os processadores que utilizamos no dia-a-dia, tais como os da Intel e AMD, são do modelo CISC. De modo mais abrangente, a arquitetura de **Harvard** normalmente possuem processadores **RISC** enquanto que as máquinas de **Von Neumann** são normalmente **CISC**. Ainda com relação a uma visão geral, podemos exemplificar arquiteturas Harvard como os processadores 8086, 8088, assim como os PICs, por outro lado, processadores 8085, Z80 constituem os modelos CISC de Von Neumann. Cabe finalmente uma diferenciação entre microprocessador, comumente abreviador por processador e microcontrolador. Os nossos computadores do dia-a-dia possuem microprocessadores, que seguem 1 dos modelos estudados acima. Damos o nome de microcontrolador ao microprocessador que, além da estrutura básica descrita acima, possua também circuitos eletrônicos auxiliares, tais como conversor analógico digital, e conversor digital analógico. A linha PIC fabricada pela Microchip, constitui um conjunto de microcontroladores, enquanto os PIC constituem clássicos exemplos de microcontroladores de baixíssimo custo e que podem ser facilmente programados em linguagem de alto nível praticamente idênticas a linguagem C.

- **Exercício**:

Tomando como base uma função em linguagem C que recebe como parâmetros um vetor de Int que corresponde a um sinal digitalizado e o seu tamanho, aumente a amplitude do sinal até o máximo possível com a restrição de que os valores devem estar na faixa de -32768 até 32767.Discuta, com base na implementação realizada, qual seria a taxonomia mais propicia para executar o referido código.

- **Solução**:



No programa acima, temos dois trechos principais: **A** e **B**. O trecho A não se beneficia consideravelmente de uma execução paralela pois encontrar o maior valor do vetor implica uma comparação integral com os seus elementos. Por outro lado, o trecho B certamente se beneficia de uma execução paralela pois uma mesma instrução, a multiplicação, é realizada para cada elemento do vetor. Dessa forma, considerando A e B, podemos dizer que o programa todo, visto como uma entidade única, se beneficia do paralelismo, sendo que a taxonomia SIMD, de acordo com a classificação de Flynn é a ideal. Tal taxonomia corresponde normalmente à arquitetura Harvard e maquinas Harvard geralmente atuam com processadores RISC. Essas características são perfeitas pois disponibilizam o paralelismo desejado que não é degradado pelas instruções RISC pois multiplicação é uma instrução básica.

- **Exercício**:

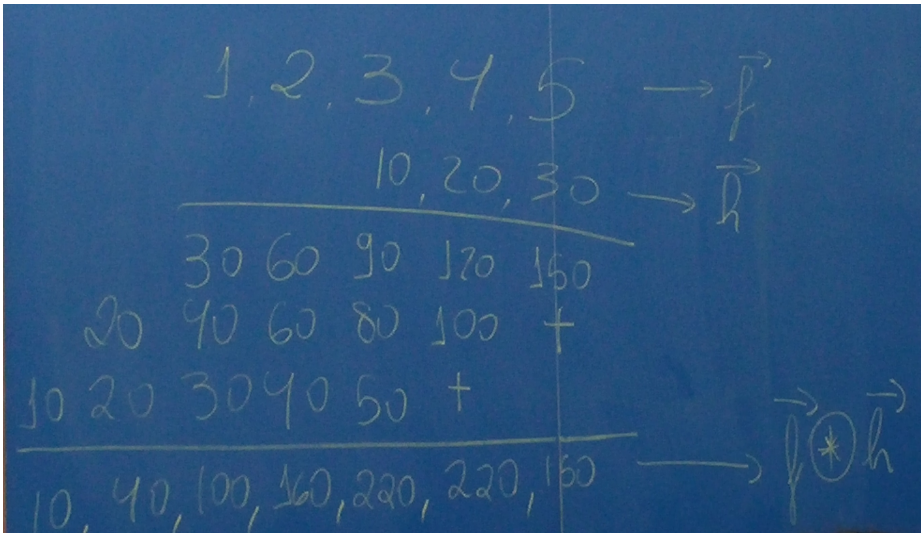
Considere o trecho de código a seguir que implementa a convolução de dois sinais discretos. Discuta a taxonomia ideal para tal.

- **Solução**:

O procedimento de convolução entre vetores discretos, tem o significado físico de filtragem, que objetiva remover frequências. Dessa forma, quando queremos remover determinada faixa de frequência de um sinal, devemos projetar um filtro, que nada mais é que um pequeno vetor de números e realizar a convolução do sinal a ser filtrado com o filtro. A convolução entre dois sinais produz um terceiro sinal que corresponde a um sinal filtrado, isto é sem as frequências indesejadas. Considerando que o sinal a ser filtrado é o vetor `f` e o filtro é `h` então a convolução de `f` com `h` é:

gif latex

Na pratica implementamos a convolução, seguindo um procedimento muito simples, conforme segue:



O vetor resultante da convolução de `f` com `h` que chamaremos de `y`, possui tamanho tal que
gif latex2
. Desse modo, determinar `y` implica em preencher um vetor com a referida função. Cada um desses elementos de `y` corresponde a uma soma de outros elementos produzidos pelo processo de cálculo da convolação.
A estrutura geral da implementação dos dois laços for conforme o rascunho que segue:

```
for(i....
    for(j....
        y[i] = .....
```

Claramente observamos que o algoritmo se beneficia de uma implementação paralela. Chegamos assim a uma situação similar à do exemplos anterior isto é, SIMD é a melhor taxonomia, implicando possivelmente numa arquitetura

Harvard com processador RISC. De modo geral, grande parte das operações envolvendo vetores, matrizes e analises relacionadas com processamentos de sinais diversos se beneficiam da referida estrutura de Hardware

---

MINI PROVA 6

- DESCREVA COMO DEVE SER OU IMPLEMENTE UM PEQUENO PROGRAMA PARA TRIANGULARIZAR UMA MATRIZ QUADRADA NXN, REALIZANDO O PIVOTEAMENTO DE GAULS. DISCUTA AS MELHORES CARACTERÍSTICAS DE HARDWARE PARA EXECUTAR ESSE ALGORITMO.

Aula 8
