Unidade Central de Processamento

A parte mais importante de um computador é a Unidade Central de Processamento, ou UCP, ou, do inglês, CPU. A CPU é responsável não apenas por executar os programas contidos na memória, mas também de controlar todos os dispositivos de entrada e saída. Seu avanço ao longo dos anos tem permitido que programas fossem executados cada vez mais rapidamente. Hoje existem processadores de vários núcleos capazes de executar várias atividades ao mesmo tempo.

Os programas são sequências finitas de passos que foram definidas por um programador para alcançar um objetivo específico. Denomina-se cada passo do programa de instrução. Não necessariamente, uma instrução escrita em uma linguagem de alto nível, como C, Java, Python, por exemplo, é diretamente transformada em uma instrução de máquina e armazenada em memória para execução da CPU. Na verdade, geralmente, uma instrução de uma linguagem de alto nível embute vários comandos e ações a serem executadas pela CPU. Essa é a principal razão da criação dessas linguagens de alto nível.

Depois de compilado, o programa de linguagem de alto nível é transformado em um programa apenas com instruções de máquina. Cada instrução de máquina contém apenas uma única operação a ser realizada pela CPU. Para ser executado, esse programa deve ser transferido para a Memória Principal. No princípio, um Operador de Máquina, copiava todas as instruções para a memória de maneira quase que manual. Hoje essa operação é realizada pelo S.OP. (Windows, Linux etc.). Assim que um usuário *clica* com o mouse, ou pressiona a tecla **Enter** do teclado solicita que um determinado programa execute, o S.Op. copia o programa para a memória e solicita que a CPU o execute.

A memória do computador apenas armazena números binários. Logo, um programa em linguagem de máquina é formado por instruções em binário. A cada instrução trazida da memória, a CPU lê seu código binário de operação para saber do que se trata, e inicia o processo de execução. Dependendo da operação, que pode ser de movimentação de dados, uma operação lógica, ou aritmética, ou uma operação de armazenamento de dados, a CPU envia ordens para que os outros dispositivos do computador atuem de forma a completar a operação. Essas ordens são enviadas através de sinais elétricos passados por dutos no computador. Os dutos são os **Barramento de Controle**.

Software x Hardware

O computador é composto por dois elementos, o software e o hardware. A diferença está na concepção. O hardware é concebido em chip, utilizando transistores interconectados. Uma vez elaborado, o hardware não pode mais ser modificado. Ele é uma solução rígida (do inglês, Hard) para o problema. Já o software é elaborado para ser armazenado numa memória e ser executado com um processador de propósito geral. Ele é uma solução flexível (do inglês, Soft) para o problema, já que o programador pode, a cada momento, modificar seu programa a fim de torná-lo cada vez melhor.

Soluções em software são sempre mais lentas do que soluções equivalentes em hardware. Isso porque para executar um programa, cada instrução deve antes ser armazenada em memória, transferidas para a CPU (memórias são muito mais lentas do que CPUs) e, só então, ser executada pela CPU. Já as soluções em hardware não utilizam instruções, elas executam as operações diretamente.

Por outro lado, as soluções em software ganham em flexibilidade, já que os programas podem ser facilmente modificados. Já as soluções em hardware, não. Uma vez concebido, um hardware não pode mais ser modificado, ele deve ser descartado para dar lugar a uma versão mais nova. Isso torna projetos em hardware muito mais caros.

Para entender melhor, podemos citar alguns exemplos de implementações em hardware comumente utilizadas. Todas são escolhidas devido ao seu caráter de pouca necessidade de modificação, mas muito demanda por alto desempenho. Por exemplo, chips de criptografia para celulares (geralmente smartphones), processadores aritméticos para acelerar os cálculos, aceleradores gráficos para gerar gráficos mais rápidos, alguns chips para fazer edições rápidas em fotos, geralmente acopladas às câmeras digitais. As implementações são feitas em software quando a demanda por desempenho não é tanta, ao mesmo tempo em que as atualizações são frequentes, como os Sistemas Operacionais, os jogos e aplicativos em geral.

Apesar de não ser tão rápida quanto gostaríamos a CPU é uma solução muito boa por permitir a execução de, praticamente, qualquer tipo de programa, se tornando uma máquina de propósito geral.

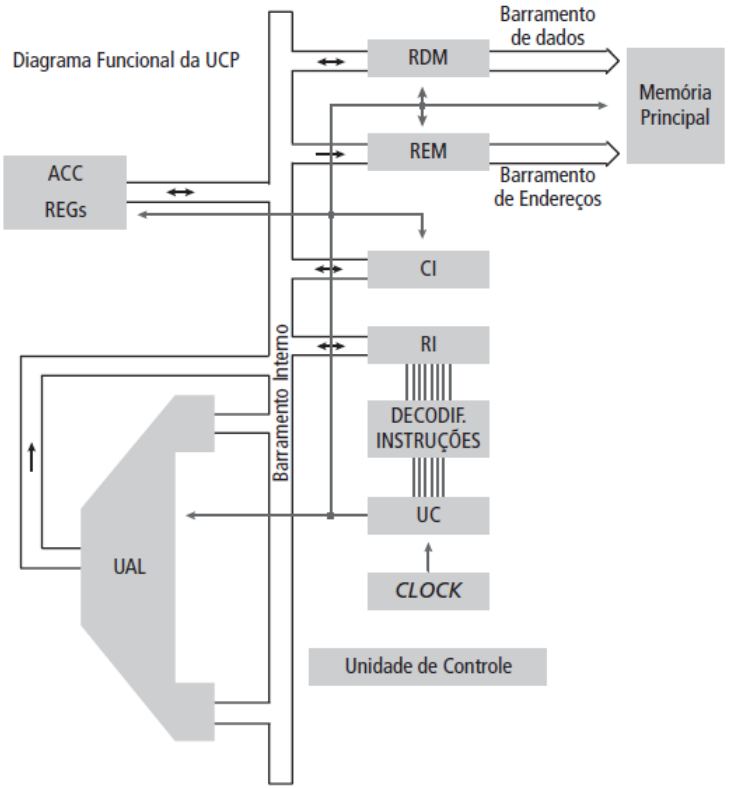
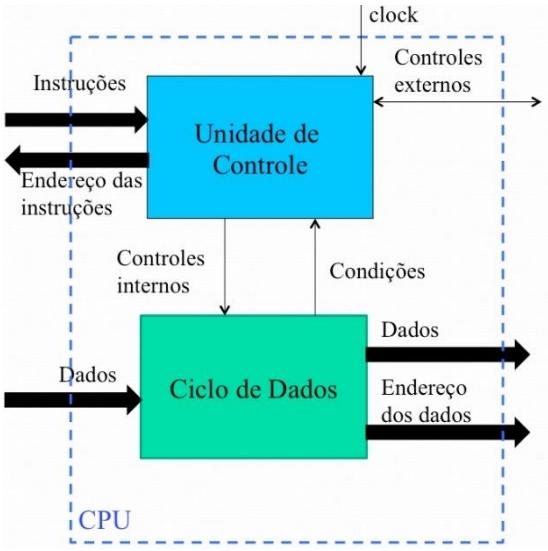


Diagrama Funcional de CPU

Estrutura de uma CPU

Toda CPU é formada por duas unidades, como podem ser vistas na figura abaixo: Unidade de Controle - UC e Unidade de Ciclo de Dados – UCD.



Estrutura de uma CPU

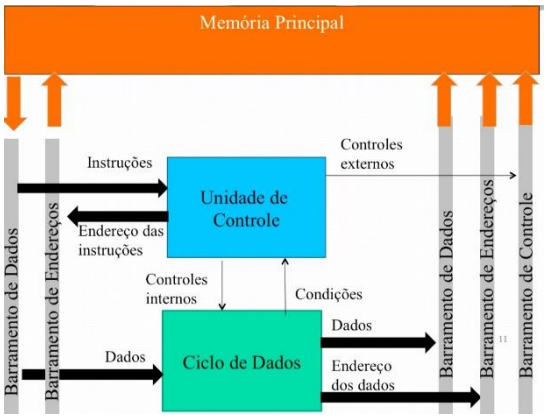
A Unidade de Controle é responsável por receber instruções pelo Barramento de Instruções. As instruções vêm da memória de acordo com o endereço enviado pela UC para a memória através do Barramento de Endereço das Instruções (à esquerda da UC na figura acima). Já Unidade de Ciclo de Dados, como o próprio nome deixa entender, é responsável por tratar os dados propriamente ditos. A Unidade de Controle não executa as instruções. Ela as lê, decodifica e passa os comandos para a UCD determinando como as instruções devem ser executadas e com quais dados. Baseada nesses comandos, a UCD pode ir buscar os dados necessários na memória, executar as devidas operações e enviar o resultado de volta para a memória para ser armazenado. Tudo controlado de acordo com os comandos internos enviados pela Unidade de Controle, que por sua vez se baseia na instrução decodificada. Os

dados lidos, ou enviados para a memória, são transmitidos através do Barramento de Dados. Os endereços são enviados para a memória através do Barramento de Endereço.

Tudo isso é controlado por um sinal síncrono de relógio (clock, do inglês). A cada batida do relógio a unidade sabe que deve executar um passo, passar os dados para quem deve, e se preparar para o próximo passo. Quanto mais rápido é o relógio mais operações por segundo o processador consegue executar e mais rápido pode se tornar. A velocidade do relógio é medida em frequência, utilizando a unidade Hertz (abreviatura é Hz). Um Hertz significa um passo por segundo. Os processadores atuais trabalham na faixa dos poucos GHz (Giga Hertz), entre 1 GHz e 5 GHz. Um Giga Hertz significa um bilhão de pulsos por segundo. É por isso que os computadores são tão incríveis. Eles não executam operações extraordinárias. Pelo contrário. Executam operações extremamente simples, como somas, subtrações e multiplicações, mas fazem isso numa velocidade incrível.

Função dos Barramentos e da Memória

Saindo um pouco de dentro da CPU, enxerga-se os barramentos e a Memória Principal, como é apresentado na figura abaixo. Para facilitar a visualização, os Barramentos de Dados e de Endereço são apresentados replicados, tanto do lado esquerdo, quanto do direito da figura.



Estrutura de uma CPU com barramentos

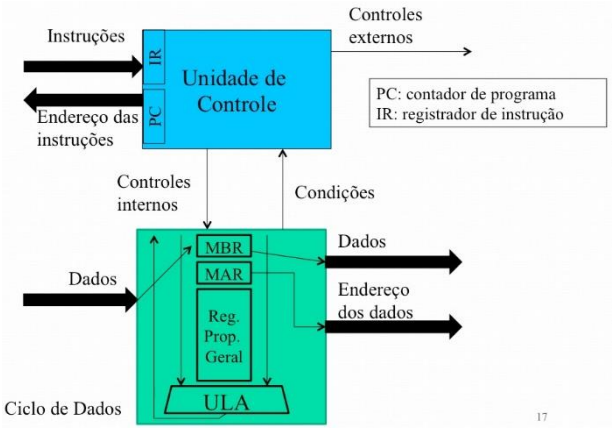
A comunicação da Unidade de Controle e da Unidade de Ciclo de Dados é feita sempre com a Memória Principal através dos barramentos. Os endereços são transmitidos sempre via Barramento de Endereços para a memória, sempre de forma unidirecional da CPU para a memória. Quando as instruções são transmitidas da memória para a Unidade de Controle, elas utilizam o Barramento de Dados. Isso porque as instruções são tratadas pela memória como um conteúdo como outro qualquer. Ela não faz distinção entre dados e instruções. O mesmo Barramento de Dados é utilizado pela Unidade de Ciclo de Dados para receber os operandos das operações a serem realizadas e para enviar os resultados de volta para a memória.

Fica clara então a importância da Memória Principal. Todo e qualquer programa só poderá ser executado a partir dela. Quando você, por exemplo, deseja executar um programa de um pen drive conectado pela USB do computador, ele antes precisa ser copiado para a Memória Principal. Só então ele será executado. A memória precisa ser grande o bastante para armazenar a maior quantidade possível de programas, e também precisa ser rápido o suficiente para buscar os dados e enviá-los o mais rapidamente possível à CPU, e também salvá-los no menor tempo possível. A velocidade das memórias é determinada essencialmente pela tecnologia de transistores utilizada. Essa tecnologia é relacionada ao preço. Quanto mais rápidas, mais caras elas são.

Registradores

Os registradores são memórias elaboradas com o mínimo de transistores possível, utilizando o que há de mais moderno em tecnologia de armazenamento. Elas são as memórias mais rápidas que podem ser construídas e por isso são também as mais caras. Por isso, aparecem numa quantidade muito pequena em um computador, na casa de alguns KBytes. Geralmente divididos em dois grupos: registradores de propósito geral e de propósito específico. Os primeiros podem ser utilizados pelos programas para quaisquer objetivos, já o outro grupo são específicos para algumas tarefas. Por exemplo, há um registrador na CPU para controlar se o processador deve continuar em execução, ou entrar em

modo de espera por nova ordem. Se esse registrador receber um valor diferente de zero, o processador entrará em modo de espera, até que receba a ordem de modificar esse valor.



Estrutura de uma CPU com registradores

Na figura os registradores de propósito específico apresentados são:

- **PC** - Program Counter: Contador de Programas
- **IR** - Instruction Register: Registrador de Instrução
- **MAR** - Memory Address Register: Registrador de Endereço
- **MBR** - Memory Buffer Register: Registrador de Dados

O PC contém o endereço de memória que será utilizado para buscar a próxima instrução a ser executada pela CPU. Antes de executar qualquer instrução, a CPU envia o conteúdo de PC para a memória através do Barramento de Endereço, a memória envia o conteúdo da memória nesse endereço através do Barramento de Dados. Esse conteúdo é então armazenado no IR.

Já o IR, que recebeu a instrução que veio da memória, tem o objetivo de guardar a instrução e passá-la para a Unidade de Controle, que é quem vai lê-la e tomar as decisões necessárias para que ela seja executada pela Unidade de Ciclo de Dados. Por se tratarem do processo de busca de instruções, o PC e o IR ficam instalados na Unidade de Controle. O PC possui conexão direta com o Barramento de Endereços, e o IR com o Barramento de Instruções.

Com relação ao MAR e ao MBR, eles possuem funções análogas ao PC e IR, respectivamente, mas referentes a dados e não a instruções. Quando uma operação precisa ser realizada com algum dado que está na memória (e não em um registrador), o endereço desse dado é passado para o MAR. A CPU então passa o conteúdo de MAR para a memória através do Barramento de Endereço, que retornará o conteúdo da memória nesse endereço através do Barramento de Dados.

O conteúdo trazido pela memória será armazenado em MBR. Só então o dado poderá ser utilizado para o processamento inicialmente planejado. O MBR e MAR possuem, respectivamente, conexões diretas com os Barramentos de Dados e de Endereços. Ambos são situados na Unidade de Ciclo de Dados, por serem utilizados nas fases de processamento das instruções.

O tamanho e quantidade dos registradores de uma CPU é uma das principais decisões de projeto. Se forem grandes demais, ou em quantidade maior do que a necessária, podem resultar em desperdício e aumento no custo do processador. Já se forem pequenos, ou em pouca quantidade, vão tornar o computador muito mais lento do que o desejado. Encontrar o tamanho e a quantidade ideais é trabalhoso e geralmente é feito através de simuladores e de muitos testes e anos de experiência.

Os registradores de propósito geral são usados para guardar as variáveis dos programas. Como eles estão presentes em quantidade reduzida, são poucas as variáveis que ficam armazenadas. As demais ficam na Memória Principal. Quando uma operação é realizada e seus dados estão nos Registradores de Propósito Geral, a CPU não precisa buscá-los na memória e o processamento torna-se muito mais rápido. A CPU tenta manter as variáveis mais utilizadas nos registradores. Ela faz isso guardando aquelas mais usadas nas últimas operações. Nem sempre funciona, mas é a melhor solução.

Unidade Lógica e Aritmética - ULA

A Unidade Lógica e Aritmética, ou ULA, se assemelha muito com uma calculadora convencional. Ela executa operações lógicas e aritméticas.

As ULAs modernas executam operações tanto com inteiros, como com números reais. A ULA recebe como entrada dois diferentes dados que são trazidos para ela dos registradores (de propósito geral, ou específico) figura acima. Unidade de Controle decide que registradores passarão seus dados para a ULA baseada no tipo da instrução que está executando. A Unidade de Controle também envia para a ULA qual operação será realizada (**soma**, **multiplicação**, **divisão**, **AND**, **OR** etc.). Assim que isso é feito, a ULA executa a operação e gera um resultado na sua saída. Esse resultado também é passado para o registrador escolhido pela Unidade de Controle, baseado na instrução em execução.

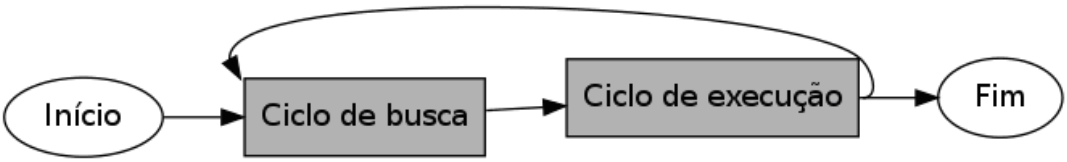
Unidade de Controle - UC

A Unidade de Controle, ao receber a instrução que está armazenada em IR, a decodifica e envia os sinais de controle para onde for necessário. Decodificar nada mais é do que ler um código em binário e interpretar a operação relativa a esse código. Dependendo da operação, os sinais de controle podem ser internos, por exemplo, para a ULA executar uma soma, ou para o conteúdo de um registrador ser transferido para a ULA. Ou pode ser externo, para um dispositivo de entrada e saída, por exemplo, ou mesmo para a Memória Principal. Tudo isso depende da instrução a ser executada.

Na próxima seção será apresentada a execução de instruções em mais detalhes, o que facilitará o entendimento do funcionamento das CPUs.

Ciclo de Instrução

Toda CPU trabalha em dois ciclos principais, o Ciclo de Busca e o Ciclo de Execução, como pode ser visto na figura abaixo. Assim que o computador é iniciado, a CPU entra no Ciclo de Busca, em seguida passa para o Ciclo de Execução e depois volta para o Ciclo de Busca. Ela continua nesse processo até que precise ser desligada, saindo do Ciclo de Execução para o estado final.



Ciclo de Instrução

Durante o Ciclo de Busca, é a Unidade de Controle que atua. Uma nova instrução é buscada da Memória para que possa ser decodificada. Nessa fase os registradores [PC] e [IR] são utilizados, como apresentados na seção anterior. O PC é logo lido para se saber que instrução será executada, essa instrução é trazida para o IR e, finalmente, é decodificada pela Unidade de Controle. Assim que esse processo termina, caso a instrução não diga respeito à um laço, ou à uma repetição, o conteúdo de PC é incrementado. Ou seja, PC recebe PC + 1. Assim, no próximo Ciclo de Busca a instrução do endereço seguinte será carregada da memória e executada. Esse comportamento garante a característica de execução sequencial dos programas.

No passo seguinte a CPU entra em Ciclo de Execução. Nessa etapa atua a Unidade de Ciclo de Dados. Agora a Unidade de Controle já sabe exatamente que operação será executada, com quais dados e o que fazer com o resultado. Essa informação é passada para a ULA e os registradores envolvidos. Durante o Ciclo de Execução há cinco possíveis tipos de operação que podem ser executadas:

- **Processador e Memória:** trata simplesmente da transferência de dados entre CPU e memória principal;
- **Processador e Entrada e Saída:** diz respeito à transferência de dados entre a CPU e um dispositivo de Entrada e Saída, como teclado, mouse, monitor, rede, impressora etc.;
- **Processamento de Dados:** são operações simplesmente de processamento dos dados, como operação aritmética ou lógica sobre os registradores da CPU;
- **Controle:** são instruções que servem para controlar os dispositivos do computador, como para ligar um periférico, iniciar uma operação do disco rígido, ou transferir um dado que acabou de chegar pela Internet para a Memória Principal;
- **Operações compostas:** são operações que combinam uma ou mais instruções das outras em uma mesma operação.

Busca de Dados

Em operações entre Processador e Memória, é necessário que dados sejam trazidos da memória para servirem de entrada para a ULA, e/ou o resultado seja levado para armazenamento na memória no final da execução. Para isso acontecer, é executada uma Busca de Dados. Isso é determinado durante a decodificação da instrução, no ciclo de Busca de Instrução. Isso acontece quando um dos parâmetros de uma operação aritmética é um endereço de memória, e não um valor diretamente, nem um registrador. Para isso, parte do conteúdo de [IR] é transferido para o [MAR]. Essa parte é justamente o endereço do parâmetro da instrução. Em seguida a Unidade do Controle requisita à memória uma leitura. Assim, o endereço, agora em MAR, é passado para a memória e o conteúdo lido da memória é passado para o [MBR]. Agora o conteúdo é transferido para a ULA para que a operação seja executada (lógica ou aritmética).

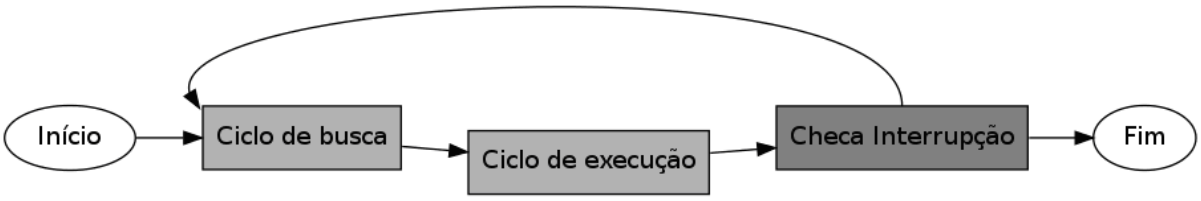
Se a instrução tiver 2 ou mais parâmetros de memória, serão necessárias outras Buscas de Dados. Como a memória é sempre mais lenta do que a CPU, instruções que necessitam Buscas de Dados são muito mais lentas do que instruções de Processamento de Dados.

Perceba que cada instrução pode exigir mais tempo de execução do que outras. Isso depende de quantos acessos à memória ela exigirá. Quanto mais acessos à memória, mais lenta a instrução. O ideal é sempre usar registradores. Mas nem sempre é possível utilizar registradores. Eles estão sempre em poucas quantidades e em menores tamanhos. Principalmente por serem caros. O que os computadores sempre tentam fazer é passar os dados da memória para os registradores assim que puderem, para que as próximas instruções sejam aceleradas.

Interrupções

Além do ciclo básico de instrução apresentado anteriormente, a CPU pode ainda executar outro tipo de tarefa. Ela diz respeito ao processamento de pedidos oriundos dos dispositivos de Entrada e Saída. Como o Ciclo de Instrução da CPU que vimos até o momento é fechado, ou seja, a CPU sempre fica em estado de repetição até que seja desligada, ela não pode atender a nenhum evento externo que não seja a execução de um programa. Por exemplo, quando um usuário pressiona uma tecla do teclado, ou faz um movimento com o mouse, ou mesmo, quando uma mensagem chega pela Internet através da placa de rede. O que a CPU deve fazer? Se ela estiver em um Ciclo de Instrução fechado como mostrado anteriormente, nada. Ela precisa parar o que está fazendo para atender ao evento ocorrido e, só então, voltar ao Ciclo de Instruções. Esse processo de parar o Ciclo de Instrução para atender a um evento externo é chamado de Interrupção.

O Ciclo de Instrução pode ser visto modificado na figura abaixo para atender às Interrupções. Todas as interrupções são recebidas e armazenadas internamente pelo dispositivo Gerenciador de Interrupções. Esse dispositivo é um chip, semelhante a uma CPU, mas bem mais simples.



Ciclo de Instruções com interrupções

Na maioria dos computadores eles vêm soldados na Placa-Mãe, mas podem também vir dentro do chip da CPU. Toda interrupção possui um código de identificação. Sempre que uma nova interrupção chega nesse gerenciador, ele armazena esse código em sua memória e manda um sinal para CPU através do Barramento de Controle. Durante seu Ciclo de Instrução, sempre que uma instrução é executada, antes de voltar para o Ciclo de Busca, a CPU checa se algum sinal de interrupção foi enviado pelo Gerenciador de Interrupção.

Quando não há uma interrupção, a execução volta ao Ciclo de Busca e o programa em execução continua a ser executado. Mas se houver uma interrupção, a CPU agora vai parar a execução do programa atual para atender a interrupção.

Por exemplo, vamos supor que o usuário tenha pressionado uma tecla do teclado. O código armazenado pelo Gerenciador de Interrupção indica que a interrupção veio do teclado. A CPU para sua

execução do programa anterior e vai iniciar a execução de um programa especial, o Tratador de Interrupção. O código do dispositivo (aqui seria o teclado) serve para a CPU saber o endereço do Tratador de Interrupção que ela vai buscar da memória. Então, ao sair da Checagem de Interrupção, a CPU muda o endereço do PC para o endereço do Tratador de Instrução.

Assim, no Ciclo de Busca a próxima instrução a ser trazida da memória e posteriormente executada será a do tratador do teclado.

Cada tipo de interrupção precisa de um tratamento específico a ser feito. No caso do teclado, o tratador vai checar que tecla foi pressionada. Isso é feito através de uma leitura à memória do teclado (sim, todos os dispositivos possuem uma pequena memória) para saber que tecla foi pressionada.

Dependendo da tecla, uma operação diferente será executada. Geralmente, a CPU adiciona o código da tecla pressionada num endereço específico de memória. Cada programa, lendo essa informação, tomará sua própria decisão sobre o que deve ser feito. O que acontece é que apenas o programa ativo no momento, vai ler esse conteúdo, executar a ação da tecla e limpar essa área de memória. Se o programa for um editor de texto, por exemplo, o código pode representar escrever a letra pressionada na posição atual do cursor dentro do texto.

Quando esse processo encerra, o tratamento é encerrado, e a CPU deve voltar à execução do programa que havia sido interrompido. Isso só é possível porque, antes de passar à execução do Tratador de Interrupção, a CPU salva os conteúdos de todos os registradores da CPU (inclusive o PC e o IR). Então, antes de devolver a execução para o programa, a CPU restaura todos os valores dos registradores antes salvos. Dessa forma, o programa retoma exatamente do ponto em que parou.

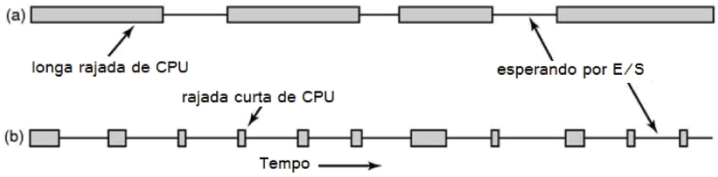
As interrupções também ocorrem se o próprio programa em execução executar uma operação ilegal. Isso é feito para evitar que a CPU entre em erro. Por Ex., se um programa tentar acessar uma área da memória que é proibida para ele, como a área de outro programa ou do S. Op. Nesse caso, o programa é interrompido e não volta mais a executar, é finalizado e a execução é devolvida ao SOp.

Algo semelhante ocorre em caso de defeitos em alguns dispositivos. Exemplo, se um programa estiver lendo um arquivo que está em um pen drive, e se for removido subitamente, uma interrupção é lançada e o programa é encerrado, já que ele não faz mais sentido estar em execução.

Desempenho

É possível agora perceber que o desempenho das CPUs depende de muito outros fatores além da velocidade do seu clock. O computador precisa ter memórias rápidas para reduzir o tempo dos Ciclos de Busca, precisam de mais registradores para usar menos a memória e também que poucas interrupções ocorram. Cada vez que uma interrupção ocorre, o programa deve ser interrompido e a chamada deve ser atendida. Isso vai atrasar demais o tempo de execução dos programas, dando a impressão de baixo desempenho.

Basicamente, há 2 tipos programas, os orientados à CPU e os orientados a Entrada e Saída. Na figura abaixo o comportamento dos primeiros é mostrado na parte (a) e o dos segundos na parte (b).



Execução com várias interrupções

Quando um programa é orientado à CPU, há momentos longos de processamento de CPU e curtos momentos de espera por um evento de Entrada e Saída. Programas que fazem muitos cálculos matemáticos, como ferramentas de simulação, projetos de engenharia, computação gráfica e planilhas de cálculos. Inicialmente os dados de entrada são passados por um dispositivo de entrada, há longos momentos de cálculos e depois os resultados são passados para um dispositivo de entrada e saída.

Em programa orientado à Entrada e Saída (b), ou interativos, há muitos momentos de interação e uso de Dispositivos de E/S, e poucos momentos de uso de CPU. Esse é o caso de programas que utilizam muito o mouse e o teclado, como os jogos e a própria navegação na internet. O que deve ser observado é que o desempenho de um computador está ligado ao perfil de cada usuário.

Os Sistemas Operacionais são os responsáveis por escolher que tarefa colocar para executar a cada momento e por quanto tempo ela deve executar até que uma nova tarefa entre em execução.

Assim, o papel do S.Op. também é fundamental e determinante no desempenho do sistema. O que ele tenta fazer no máximo que pode, é tentar ocupar os tempos de espera de um programa com a execução de outro. Tarefa nada fácil!

Exemplo de execução de um programa

Suponha que queiramos executar uma instrução de máquina que soma dois números que estão na memória e salve o resultado em outro endereço de memória. Para tal, vamos indicar que a memória (M) se comporta como um vetor (um array) e entre colchetes indicaremos o endereço do dado, ou da instrução. Sendo assim, a instrução que gostaríamos de executar seria:

200: M[100] = M[101] + M[102]

Nesse caso, vamos ler que no endereço **200** da memória há uma instrução que precisa somar o conteúdo do endereço **101**, com o conteúdo do endereço **102** e salvar o resultado no endereço **100** da memória. Supondo que **M[101]** contenha o valor 10, **M[102]** contenha o valor **20**, ao final da execução, o endereço **100** de memória (**M[100])** deverá conter o valor **30**.

Como uma instrução como essa será executada depende de cada arquitetura. Aqui vamos utilizar uma abordagem que quebra as instruções em pequenos passos simples, que facilitam o trabalho de decodificarão da CPU. Sendo assim, esse programa seria transformado na seguinte sequência de instruções e executado.

PC=200;

//Envia Comando de leitura de instrução para a memória

IR <- (M[100] = M[101] + M[102]) // Busca instrução da memória

PC = PC + 1

// Instrução é passada do IR para a Unidade de Controle

A primeira ação seria realizar o **Ciclo de Busca**, visando trazer a instrução a ser executada da memória para o processador. O endereço da instrução (**200**) seria passado para o **PC** e um comando de leitura de instrução seria passado para a memória. Baseada no endereço trazido por **PC**, a memória localizaria a instrução e a enviaria para o processador, que a armazenaria no registrador **IR**. Antes de passar a instrução para a Unidade de Controle para dar início à execução, o registrador **PC** é atualizado para o próximo endereço de memória, no caso, **201**.

O próximo passo será iniciar o **Ciclo de Execução**:

// O primeiro dado é trazido da memória para o registrador R1

MAR = 101

// Envia comando de leitura de dado para a memória

MBR <- 10

R1 = MBR // valor lido da memória é passado para o registrador R1

Como os dados a serem operados estão também na memória, é antes necessário executar uma operação de **Busca de Operando**, ou **Busca de Dado**. O primeiro operando está no endereço **101**. Sendo assim, o endereço **101** é passado para o registrador de endereço (**MAR**). Esse endereço é passado para a memória e é enviado um comando de leitura de dado. O conteúdo, o valor 10, é então localizado pela memória e enviado para o processador, que o armazena no registrador de dados (**MBR**). Como o **MBR** será utilizado nas próximas etapas de execução, seu conteúdo é salvo em um registrador de propósito específico, o **R1**.

Em seguida, a Unidade de Controle busca do segundo operando, contido no endereço **102**:

// O segundo dado é trazido da memória para o registrador R1

MAR = 102

// Envia comando de leitura de dado para a memória

MBR <- 20

R2 = MBR // valor lido da memória é passado para a memória o registrador R2

Essa etapa ainda faz parte do **Ciclo de Execução**, e também diz respeito a uma **Busca de Dado**. A busca é mesma do passo anterior, mas agora o endereço buscado é o **102**, e o conteúdo é o **20**, que é repassado para o registrador **R2**.

O próximo passo do **Ciclo de Execução** é executar a operação aritmética propriamente dita. Isso geralmente é feito entre registradores de propósito geral, por serem mais rápidos do que se fosse tratar dados da memória. Os conteúdos de **R1** e **R2** são somados e armazenados em **R3**:

R3 = R1 + R2

Para finalizar o processo, o resultado deve ser armazenado de volta na memória:

MAR = 100 // Endereço é passado para MAR

MBR = R3 // Resultado da operação é passado para MBR

// Comando de escrita é passado para a memória

M[100] <- 30 // Endereço 100 da memória recebe o valor 30

Para ser realizado, é preciso executar uma operação de escrita na memória. O endereço **100** é então passado para **MAR** e o resultado da operação, salvo em **R3** é passado para **MBR**. Quando o comando de escrita é enviado pela Unidade de Controle para a memória, ela lê o endereço **100** pelo Barramento de Endereço e o valor **30** pelo Barramento de Dados e salva, o valor **30** no endereço **100**.

Com isso a operação é finalizada. Essa operação foi executada em aproximadamente 14 passos. Esse valor é aproximado porque alguns deles são apenas o envio de sinal para a memória, e isso geralmente é feito em paralelo com o passo seguinte. Se cada passo for executado dentro de uma batida do relógio (ou **ciclo de clock**), teremos 14 ciclos de clock para uma única instrução. Mas perceba que o acesso à memória é sempre mais lento do que a execução do processador. Se cada acesso à memória levar 3 ciclos de clock, tem-se um total de 20 ciclos de clock.

Apenas a memória tipo Cache poderia ser acessada com apenas 3 ciclos de clock. A memória principal convencional precisa de entre 10 e 15 ciclos para ser lida. Algumas instruções podem levar muito mais ciclos do que isso, como operações com Ponto Flutuante (números reais), ou de acesso a um periférico, como o disco rígido. Isso depende muito de como o projeto do computador é elaborado.

Apesar de o computador parecer pouco efetivo na execução de uma simples soma, como ele executa numa frequência de clock muito alta, ele acaba executando muitas operações por segundo. Então, utilizar apenas a frequência de clock como medida de desempenho não é uma boa idéia. O mais utilizado é medir a quantidade de operações aritméticas que o processador é capaz de executar por segundo. Os PCs estão na escala dos alguns Milhões de Instruções por Segundo (ou MIPS).

Aumentando o desempenho com Pipeline

Pelo o que foi visto, até o momento, a execução de um programa é essencialmente sequencial, ou seja, uma instrução só é executada quando a anterior termina. Ao longo do nosso curso veremos que há dois modos de paralelismo que podem ser utilizados para melhorar ainda mais o desempenho do processador. O primeiro deles é através do chamado **Paralelismo em Nível de Hardware** é obtido quando são replicadas as unidades do processador para que elas funcionem em paralelo, reduzindo assim o tempo de execução dos programas. A segunda forma é através do **Paralelismo em Nível de Instruções**, ou ILP (do inglês, *Instruction Level Parallelism*). Nesse caso, as unidades do processador não são duplicadas, mas melhores organizadas para que não fiquem ociosas. Há duas formas principais de implementar o ILP, uma delas é através do **Pipeline** e a outra é através de **processadores Superescalares**. Aqui vamos tratar do Pipeline, e no capítulo sobre Processamento Paralelo, vamos tratar as outras formas de paralelismo.

Considere a divisão do Ciclo de Instrução de um determinado processador nas 5 etapas:

- **Carregar instrução FI:** Traz a instrução da memória para o processador, armazena em IR (essa etapa também é chamada de *Fetch de Instrução*) e a decodifica para execução no passo seguinte.
- **Carregar operandos FO:** Traz os operandos da operação dos registradores para a ULA, para que a operação seja realizada sobre eles, também chamada de *Fetch de Operandos*.
- **Executar instruções EI:** Executa operação lógica ou aritmética propriamente dita.
- **Escrever em memória WM:** Escreve o resultado da operação em memória, se necessário.
- **Escrever em registrador WR**: Escreve o resultado da operação em um dos registradores, se necessário.

Esse é um dos Ciclos de Instrução mais simples que poderíamos imaginar, organizado em apenas 5 etapas. Processadores convencionais, como os da Intel que usamos em nossos computadores, executam instruções em cerca de 18 etapas.

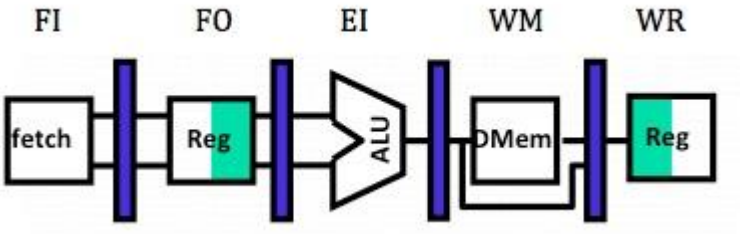
Cada instrução deve passar pelos 5 passos descritos para ser executada. Se cada etapa requerer apenas 1 ciclo de clock para ser executada, quantos ciclos são necessários para executar um programa de 20 instruções? Essa conta é simples. Cada instrução deve passar pelas cinco etapas, e cada etapa leva 1 ciclo de clock, sendo assim, o programa levará 20 vezes 5 ciclos, ou seja, 100 ciclos de clock.

Analisando o que acontece com cada etapa na medida em que o programa é executado. A primeira instrução vai passar pela etapa **FI**, que a leva para o IR e a decodifica. Em seguida ela é passada para a etapa **FO**, e os dados necessários para a operação são levados dos respectivos registradores para a ULA. Agora, observe. Neste exato momento, a segunda instrução do programa está parada na memória, aguardando sua vez para ser executada. Ao mesmo tempo, a etapa **FI** está ociosa. Por que a etapa FI não pode entrar em ação e trabalhar com a segunda instrução do programa, enquanto a primeira está na etapa FO?

O mesmo vai ocorrer com todas as etapas de execução da primeira instrução do programa. Ela vai ser executada na etapa **EI**, depois vai passar para a etapa **WM** que checará se há necessidade de copiar o resultado para a memória e, finalmente, para a **WR**, que copiará o resultado para um dos registradores. Quando a primeira instrução estiver na etapa **WR**, as etapas anteriores estarão todas ociosas. Por que não aproveitar o tempo ocioso para colocar as etapas anteriores para irem adiantando a execução das próximas instruções? É isso que propõe o Pipeline!

O Pipeline vai separar as etapas de execução de instruções em unidades físicas independentes, assim, uma etapa pode trabalhar com uma instrução, ao mesmo tempo em que uma outra unidade trabalha com uma outra instrução. É a mesma estratégia utilizada pela indústria de produção em massa para fabricar carros, por exemplo. Enquanto um chassi está sendo montado, outro está recebendo a carroceria, outro o motor, outro sendo pintado e outro recebendo o acabamento interno. Todas as etapas trabalhando em paralelo e vários carros sendo tratados ao mesmo tempo. Esta estratégia aumenta o desempenho da execução das instruções de forma grandiosa.

Na figura a seguir é apresentado adequações necessárias no processador para que as etapas possam ser organizadas em Pipeline. O processador apresentado trabalha com 5 Estágios de Pipeline.



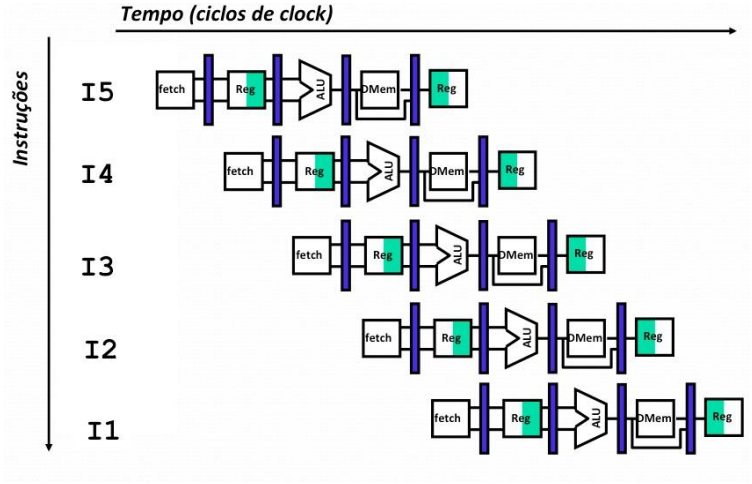
Processador adaptado para trabalhar com Pipeline de cinco estágios

A primeira mudança necessária é a separação da memória em duas partes independentes (ou 2 memórias mesmo). Uma parte será utilizada apenas para instruções (representadas na figura pela palavra Fetch), e outra apenas para os dados (representada por DMem). Isso é necessário para que a etapa **FI** acesse a memória para buscar a próxima instrução, ao mesmo tempo em que a **WM** acessa a memória para salvar o resultado de outra instrução anterior. Se houvesse apenas uma memória para dados e instruções, isso não seria possível. Essa mudança vai contra o que foi projetado na (Arquitetura de von Neumann), e foi considerado um grande avanço. Batizada de Arquitetura Harvard.

Outra mudança importante foi a adição de memórias intermediárias entre cada etapa. Na figura acima essas memórias são representadas pelos retângulos preenchidos e sem nenhuma palavra sobre eles. Essas memórias são utilizadas para armazenar o resultado da etapa anterior e passá-lo para a etapa posterior no ciclo seguinte. Elas são necessárias porque as etapas não executam necessariamente sempre na mesma velocidade. Se uma etapa for concluída antes da etapa seguinte, seu resultado deve ser guardado nessas memórias para aguardar que a etapa seguinte conclua o que estava fazendo. Só então ela poderá receber o resultado da etapa anterior.

Na produção de um carro Ex.: a etapa de instalação do motor pode ser mais rápida do que a de pintura Então, se um carro recebeu motor, ele deve ser guardado em local temporário até que o carro anterior tenha pintura concluída. Assim, a etapa de instalação do motor pode receber um novo carro.

Qual o benefício da execução em Pipeline? Para isso, vamos analisar a figura abaixo.



Execução em pipeline de cinco estágios

O eixo horizontal representa o tempo e no vertical as instruções a serem executadas (I1, I2, I3, I4 e I5). Na figura a instrução I1 já passou por todas as etapas e está em WR, enquanto isso, I2 está em WM, I3 está em EI, I4 está em FO e I5 ainda está em FI. Como o Pipeline possui 5 estágios, precisa no mínimo de 5 instruções para encher o Pipeline e, a partir daí, inicia-se o ganho de desempenho.

No exemplo anterior, considerando que cada etapa leve 1 ciclo de clock para ser concluída. Quantos ciclos são necessários para executar 20 instruções agora com Pipeline? No início, o Pipeline está vazio, então a instrução **I1** deve passar por todas as 5 etapas para ser concluída, levando então 5 ciclos de clock. Mas, a instrução **I2** acompanhou **I1** durante toda execução e terminou no ciclo seguinte, ou seja, em 6 ciclos de clock. Em seguida, a instrução **I3** é concluída em 7 ciclos de clock, **I4** em 8 ciclos, **I5** em 9 ciclos, assim em diante, até a conclusão da 20ª instrução, que ocorre em 24 ciclos.

Comparado com o exemplo sem Pipeline, que executou o mesmo programa em 100 ciclos, o ganho foi de 4,17 vezes. Se o programa tivesse 200 instruções, levaria 1000 ciclos de clock sem Pipeline e 204 ciclos com Pipeline, o resultaria num ganho de 4,9 vezes.

Limitações do Pipeline

Infelizmente, nem sempre o processador consegue usufruir do ganho máximo de desempenho ao usar o Pipeline. Há vários riscos que podem fazer com que o Pipeline seja interrompido e precise ser reiniciado, ou impossibilitado até de iniciar. Os riscos são:

- Riscos Estruturais
- Riscos de Dados
- Riscos de Controle

Os **Riscos Estruturais** são limitações físicas do processador. O exemplo mais simples é a separação da memória em Memória de Dados e Memória de Instruções. Se isso não ocorrer, as etapas de FI e WM não podem ser executadas ao mesmo tempo.

O **Risco de Dados** ocorre quando uma instrução depende do resultado de uma instrução anterior que está no Pipeline e ainda não está pronta. Imagine o trecho de programa a seguir:

I1:r1 = r2 + r3
I2:r4 = r1 – r3

A instrução I1 inicia primeira e logo avança nas etapas do Pipeline. Logo depois dela vem I2. Quando I2 for buscar o valor de r1 na etapa FO, ele ainda não estará pronto, porque I1 ainda está em EI. A instrução I1 precisaria concluir a última etapa (WM) para que I2 pudesse executar FO. Nesses casos, se dize que há uma *Dependência de Dados*. Isso cria uma bolha no Pipeline, o processador tem que

avançar I1 até a conclusão de WM, e parar a execução de I2 e todas instruções seguintes até então. Quando concluir I1 é que o I2 e as próximas instruções seriam liberadas para continuar a execução.

O **Risco de Controle** ocorre quando qualquer mudança no fluxo de controle do processador. Ou seja, quando a execução não sequencial. O Pipeline vale a pena quando há uma grande sequência de instruções sendo executadas. O processador confia que depois da instrução I1 ele executará a I2, depois a I3, e assim sucessivamente. Mas o que acontece, por exemplo, se o processador estiver executando a instrução I10, e essa instrução ordenar que o programa salte para a instrução I30? Isso pode ocorrer se a instrução se tratar de uma repetição, ou uma chamada a uma função.

A mudança de controle também pode ocorrer por meio de interrupção, provocada por um dispositivo de entrada e saída, ou pelo próprio Sistema Operacional, quando determina que um programa seja interrompido para passar a execução para outro. Quando há uma mudança no fluxo de execução desta maneira, todas as instruções que estão no Pipeline são removidas, o fluxo é modificado, e o Pipeline começa a ser preenchido todo novamente.

As técnicas de Pipeline evoluíram, entretanto, nenhuma técnica é capaz de evitar todas as possíveis perdas de desempenho. Até boas práticas de programação podem ajudar a otimizar a execução dos programas, e os compiladores também ajudam bastante neste aspecto.

O desempenho dos computadores

O desempenho dos processadores e dos computadores é muito valorizado pelas empresas por agregarem muito valor a elas. Como se costuma dizer, tempo é dinheiro. Então quanto menos tempo se espera para um computador realizar uma tarefa, mais tempo resta para a empresa se dedicar a outras atividades. O desempenho é tão importante, que há uma corrida silenciosa entre empresas, universidades e governos para saber quem é capaz de produzir o computador mais rápido do mundo.

A organização chamada Top 500 organiza uma competição para conhecer quem são esses campeões de desempenho e anualmente geral uma lista com os 500 computadores mais velozes. Atualmente a China e os Estados Unidos disputam o topo da lista do Top 500. Nas décadas após a Segunda Guerra Mundial os países disputavam uma guerra silenciosa (a Guerra Fria) para saber quem era o país mais poderoso em poder bélico e em tecnologias, como a corrida espacial.

Ter um computador poderoso não significa apenas ser capaz de realizar tarefa mais rapidamente, mas ser também capaz de realizar certas tarefas que seriam impossíveis em computadores menos poderosos. Um exemplo disso é a construção de um computador que haja de forma semelhante ao cérebro humano. Chegar a esse ponto significa ser capaz de construir sistemas que possam substituir o homem em várias tarefas complexas, como dirigir máquinas e até mesmo operar computadores. Outro exemplo seria simular o comportamento perfeito da reação do corpo humano a drogas. Assim, não seria mais necessário o uso de cobaias para testar medicamentos.

Acesse http://www.top500.org, site Top 500 em e conheça as super máquinas da computação.

Medidas de Desempenho

São três métricas de desempenho dos computadores: CPI - Ciclos de Clock por Instrução; MPIS - Milhões de Instruções por Segundo e MFLOPS - Milhões de Instruções em Ponto Flutuante por Segundo.

Ciclos de Clock por Instrução – CPI

Determina quantos ciclos de clock são necessários para executar uma determinada instrução. Vimos que o Ciclo de Instrução é organizado em várias etapas e que isso depende de instrução para instrução. Se uma instrução acessar mais memória do que outra, ela será mais lenta. Instruções que operam com Pontos Flutuantes são mais lentas do que as operações com números inteiros. É fácil perceber a razão disso. Operações com números reais são mais complexas de serem resolvidas, porque devem ser realizadas para a parte fracionária e para a inteira, e depois o resultado deve ser consolidado. Assim, simulações são realizadas com um processador e são calculados quantos ciclos de clock cada tipo de instrução necessita em média para ser completada. Este é o CPI.

Milhões de Instruções por Segundo - MIPS

O CPI é uma medida utilizada para medir o desempenho do processador para cada tipo de instrução, mas não é muito boa para medir o desempenho para a execução de programas, que é o objetivo de todo computador. Isso porque os programas são geralmente formados por instruções de todos os

tipos, com inteiros, ponto flutuante, acessando muita ou pouca memória. Outro fator é que fatores como, Pipeline, tecnologia de memória e tamanho da memória Cache, podem fazer com que uma instrução seja executada lenta agora, e rápida logo em seguida. Para contornar isso, uma métrica muito utilizada é o MIPS. Ela determina quantos Milhões de Instruções são executadas pelo computador a cada segundo. Nesse caso, programas que demandam muito esforço do computador são executados, a quantidade de instruções é contada e depois dividida pela quantidade de segundos da execução. Caso o CPI médio (também chamado de CPI Efetivo) do computador já tenha sido calculado anteriormente, o MIPS pode ser calculado pela fórmula:

Onde, Clock é a frequência do relógio do processador em Hertz, **CPI** é o CPI médio e **M** é um milhão (). É necessário dividir o resultado por **M** porque a medida do MIPS é sempre dada na escala de milhões. Por exemplo, se um processador de 2 GHz e CPI médio de 4 ciclos por instrução, o MIPS desse processador será: O resultado dessa operação será 0,5M, ou 500K. Isso porque 2 dividido por 4 é 0,5, e 1 Giga dividido por 1 Mega, resulta em 1 Mega.

Milhões de Instruções em Ponto Flutuante por Segundo - MFLOPS

Uma alternativa para o MIPS é o MFLOPS. O MIPS é muito eficiente, mas não para comparar programas diferentes. Para calcular o MFLOPS, são executadas apenas instruções que operam com Ponto Flutuante e são calculados quantos segundos se passaram para cada milhão delas. Assim, ela pode ser definida da seguinte forma:

A única diferença para o cálculo do MIPS é que apenas as instruções que operam com Ponto Flutuante são consideradas. Assim, diz respeito a quantos ciclos de clock em média são necessários para executar uma instrução de ponto flutuante.

Os computadores pessoais e comerciais estão na escala MFLOPS. Já os supercomputadores trabalham na escala de GFLOPS (Giga FLOPS). Aqueles computadores que lideram a lista do Top 500, e são capazes até de mudar o PIB de um país, trabalham na escala do TFLOPS (Tera FLOPS).

Exemplos de calcular o desempenho de um processador

Suponha que um programa é executado num processador de 40MHz. A Tabela abaixo apresenta os CPIs coletados para cada tipo de instrução, bem como sua quantidade de instruções para um determinado programa com 100.000 instruções.

Exemplo de configuração de um processador

Tipo de instrução	CPI	Número de instruções
Aritmética com Inteiros	1	45.000
Operações de acesso à Memória	4	32.000
Operações com Ponto Flutuante	2	15.000
Instruções de salto e desvio	5	8.000

Para este exemplo, vamos calcular o CPI efetivo, o MIPS e o MFLOPS.

O CPI Efetivo é simplesmente a média ponderada dos CPIs apresentados para o programa. Isso pode ser feito da seguinte forma:

- Para o MIPS pode ser calculado como: Ou seja, para o programa examinado, o processador teve um desempenho de 16,46 milhões de instruções por segundo. Se o objetivo for calcular o MIPS geral, e não específico para esse programa, deve-se utilizar a média aritmética de todos os CPI calculados, e não a média ponderada.
- Para calcular o MFLOPS seguimos a mesma estratégia, mas dessa vez utilizamos apenas o CPI para instruções de ponto flutuante. Ou seja, Isso significa que esse processador apresentou um desempenho de 20 milhões de instruções de ponto flutuante por segundo.

Processadores

Introdução

Os processadores (ou CPUs, de Central Processing Unit) são chips responsáveis pela execução de cálculos, decisões lógicas e instruções que resultam em todas as tarefas que um computador pode fazer e, por esse motivo, são também referenciados como "cérebros" dessas

máquinas. Embora haja poucos fabricantes (essencialmente, Intel, AMD e VIA), o mercado conta com uma grande variedade de processadores. Apesar disso e das diferenças existentes entre cada modelo, todos compartilham de alguns conceitos e características.

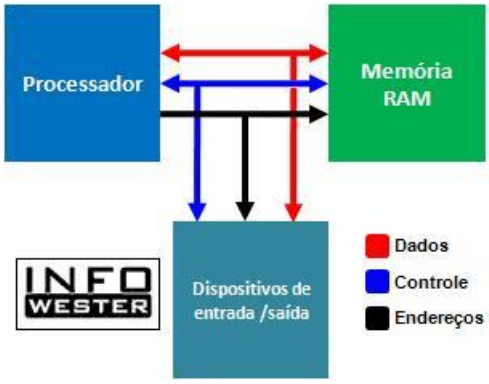
O trabalho de um processador

O processador é um chip de silício responsável pela execução das tarefas cabíveis a um computador. Para entender como um processador trabalha, é conveniente dividirmos um computador em três partes: processador, memória e um conjunto de dispositivos de entrada e saída (ou I/O, de *Input/Output*). Neste último, encontra-se qualquer item responsável pela entrada ou saída de dados no computador, como monitores de vídeo, teclados, mouses, impressoras, scanners, discos rígidos, etc. Nesse esquema, obviamente, o processador exerce a função principal, já que a ele cabe o acesso e a utilização da memória e dos dispositivos de entrada e saída para a execução de suas atividades.

Para entender melhor, suponha que se queira que o computador execute um programa qualquer. Um programa consiste em uma série de instruções que o processador deverá executar para que a tarefa solicitada seja realizada. Para isso, o processador transfere todos os dados necessários à execução, de um dispositivo de entrada e/ou saída (como um disco rígido para a memória). A partir daí, todo o trabalho é realizado e o que vai ser feito do resultado depende do programa. O processador pode ser orientado a enviar as informações processadas para o HD novamente ou para uma impressora, tudo depende das instruções.

Barramentos

A imagem a seguir ilustra a comunicação entre o processador, a memória e o conjunto de dispositivos de entrada e saída. Note-se que a conexão entre esses itens é indicada por setas. Isso é feito para que você possa entender a função dos barramentos. Ou seja, estes são os responsáveis pela interligação e comunicação dos dispositivos em um computador. Para o processador se comunicar com a memória e com o conjunto de dispositivos de entrada e saída, há 3 setas, isto é, barramentos: um se chama **barramento de endereços** (*address bus*); outro, **barramento de dados** (*data bus*); o terceiro, **barramento de controle** (*control bus*).



O barramento de endereços, basicamente, indica de onde os dados a serem processados devem ser retirados ou para onde devem ser enviados. A comunicação por esse barramento é unidirecional, razão pela qual só há seta em uma das extremidades da linha no gráfico que representa a sua comunicação. Como o nome deixa claro, é pelo barramento de dados que os dados transitam. Por sua vez, o barramento de controle faz a sincronização das referidas atividades, habilitando ou desabilitando o fluxo de dados, por exemplo.

Para você compreender melhor, imagine que o processador necessita de um dado presente na memória. Pelo barramento de endereços, ele obtém a localização desse dado dentro da memória. Como precisa apenas acessar o dado, o processador indica pelo barramento de controle que esta é uma operação de leitura na memória. O dado é então localizado e inserido no barramento de dados, por onde o processador, finalmente, o lê.

Clock interno e clock externo

Em um computador, todas as atividades necessitam de sincronização. O **clock** serve justamente para isso, ou seja, basicamente, atua como de sinal de sincronização. Quando os dispositivos do computador recebem o sinal de executar suas atividades, dá-se a esse acontecimento o nome de "pulso de clock". Em cada pulso, os dispositivos executam suas tarefas, param e vão para o próximo ciclo de clock.

A medição do clock é feita em *hertz* (Hz), a unidade padrão de medidas de freqüência, que indica o número de oscilações ou ciclos que ocorre dentro de uma determinada medida de tempo, no caso, segundos. Assim, se um processador trabalha à 800 Hz, por exemplo, significa que é capaz de lidar com 800 operações de ciclos de clock por segundo. Em termos práticos, a palavra kilohertz (KHz) é utilizada para indicar 1000 Hz, assim como o termo megahertz (MHz) é usado para indicar 1000 KHz (ou 1 milhão de hertz), ou gigahertz (GHz) para 1000 MHz, e assim por diante. Com isso, se um processador tem, por exemplo, uma frequência de 800 MHz, significa que pode trabalhar com 800 milhões de ciclos por segundo.

As frequências com as quais os processadores trabalham são chamadas também de **clock interno**. Mas, os processadores também contam com o que chamamos de **clock externo** ou **Front Side Bus** (FSB) ou, ainda, **barramento frontal**.

O FSB existe porque, devido a limitações físicas, os processadores não possam se comunicar com a memória (mais precisamente, como a ponte norte - ou *northbridge* - do chipset, que contém o controlador da memória) usando a mesma velocidade do clock interno. Assim, quando essa comunicação é feita, o clock externo, de freqüência mais baixa, é que é usado. Note que, para obter o clock interno, o processador usa uma multiplicação do clock externo. Para entender melhor, suponha que um determinado processador tenha clock externo de 100 MHz. Como o seu fabricante indica que esse chip trabalha à 1,6 GHz, seu clock externo é multiplicado por 16: 100 x 16 = 1600 MHz ou 1,6 GHz.

É importante deixar claro, no entanto, que se dois processadores diferentes - um da Intel e outro da AMD, por exemplo - tiverem clock interno de mesmo valor (ex: 2,8 GHz) não significa que ambos trabalham à mesma velocidade. Cada processador tem um projeto distinto e conta com características que determinam o quão rápido é. Assim, um determinado processador pode levar, por exemplo, 2 ciclos de clock para executar uma instrução. Em outro processador, essa mesma instrução pode requerer 3 ciclos. Além disso, muitos processadores - especialmente os mais recentes - transferem 2 ou mais dados por ciclo de clock, dando a entender que um processador que faz transferência de 2 dados por ciclo e que trabalha com clock externo de 133 MHz, o faz à 266 MHz. Por esses e outros motivos, é um erro considerar apenas o clock interno como parâmetro de comparação entre processadores diferentes.

Bits dos processadores

O número de bits é outra importante característica dos processadores e, naturalmente, tem grande influência no desempenho desse dispositivo. Processadores mais antigos, como o 286, trabalhavam com 16 bits. Durante muito, no entanto, processadores que trabalham com 32 bits foram muitos comuns, como as linhas Pentium, Pentium II, Pentium III e Pentium 4 da Intel, ou Athlon XP e Duron da AMD. Alguns modelos de 32 bits ainda são encontrados no mercado, todavia, o padrão atual são os processadores de 64 bits, como os da linha Core 2 Duo, da Intel, ou Athlon 64, da AMD.

Quanto mais bits internos o processador trabalhar, mais rapidamente ele poderá fazer cálculos e processar dados em geral, dependendo da execução a ser feita. Isso acontece porque os bits dos processadores representam a quantidade de dados que os circuitos desses dispositivos conseguem trabalhar por vez. Um processador com 16 bits, por exemplo, pode manipular um número de valor até 65.535. Se esse processador tiver que realizar uma operação com um número de 100.000, terá que fazer a operação em duas partes. No entanto, se um chip

trabalha a 32 bits, ele pode manipular números de valor até 4.294.967.295 em uma única operação. O valor é superior a 100.000, a operação será possível em uma única vez.

Processadores de 64 bits x Processadores de 32 bits

Tanto a Intel como a AMD já colocaram no mercado processadores que trabalham a 64 bits. Muita gente sabe que os modelos de 64 bits são melhores que os de 32 bits e este artigo se propõe a mostrar exatamente como e onde ocorre essas melhorias.

Processadores de 16 bits, 32 bits ou 64 bits, trata-se de bits internos do chip - em poucas palavras, isso representa a quantidade de dados e instruções que o processador consegue trabalhar por vez. Por exemplo, com 16 bits um processador pode manipular um número de valor até 65.535. Se certo número tem valor 100.000, ele terá que fazer a operação em duas partes. No entanto, se um chip trabalha a 32 bits, ele pode manipular números de valor até 4.294.967.296 em uma única operação.

Para calcular esse limite, basta fazer 2^N N bits internos do processador. Então, qual o limite de um processador de 64 bits? Ou:

$$\mathbf{2^{64} = 1.84467441 \times 10^{19}} \rightarrow \text{Um valor extremamente alto!}$$

Se for utilizado um editor de textos. É improvável que esse programa chegue a utilizar valores grandes em suas operações. Neste caso, qual a diferença entre utilizar um processador de 32 bits ou 64 bits, sendo que o primeiro será suficiente? Como o editor utiliza valores suportáveis tanto pelos chips de 32 bits quanto pelos de 64 bits, as instruções relacionadas serão processadas ao mesmo tempo (considerando que os chips tenham o mesmo clock).

Por outro lado, aplicações em 3D ou programas como AutoCad requerem boa capacidade para cálculo e aí um processador de 64 bits pode fazer diferença. Suponha que determinadas operações utilizem valores superiores a 4.294.967.296. Um processador de 32 bits terá que realizar cada etapa em duas vezes ou mais, dependendo do valor usado no cálculo. Um processador de 64 bits fará esse trabalho uma única vez em cada operação.

No entanto, há outros fatores a serem considerados. Um deles é o sistema operacional (SOp). O funcionamento do computador está diretamente ligado à relação entre o sistema operacional e o hardware como um todo. O SOp é desenvolvido de forma a aproveitar o máximo de recursos da plataforma para o qual é destinado. Assim, o Windows XP ou uma distribuição Linux com um kernel desenvolvido antes do surgimento de processadores de 64 bits são preparados para trabalhar a 32 bits, mas não a 64 bits.

A influência do sistema operacional

Um SOp de 32 bits para rodar em um computador com processador de 64 bits, o primeiro não se adaptará automaticamente e continuará mantendo sua forma de trabalho. Com isso, é necessário o desenvolvimento de sistemas operacionais capazes de rodar a 64 bits.

O Desenvolvimento ou a adaptação de um sistema operacional para trabalhar a 64 bits não é tão trivial assim. Na verdade, é necessário que o SO seja compatível com um processador ou com uma linha de processadores, já que pode haver diferenças entre os tipos existentes. Em outras palavras, o sistema operacional precisa ser compatível com chips da AMD ou com chips da Intel. Se possível, com os dois.

No caso do Windows XP, a Microsoft disponibilizou a versão "Professional x64", compatível com os processadores AMD Athlon 64, AMD Opteron, Intel Xeon (com instruções EM64T) e Intel Pentium 4 (com instruções EM64T). De acordo com a Microsoft, a principal diferença entre essa e as versões de 32 bits (além da compatibilidade com instruções de 64 bits) é o suporte de até 128 GB de memória RAM e 16 TB de memória virtual. Se a aplicação para o qual o computador é utilizado manipula grande quantidade de dados e valores, de nada adianta ter processamento de 64 bits, mas pouca memória, já que, grossamente falando, os dados teriam que "formar fila" para serem inseridos na memória, comprometendo o desempenho.

O mesmo ocorre com o Linux. Se você visitar o site de alguma distribuição para baixar uma versão do sistema operacional, muito provavelmente encontrará links que apontam para diversas versões. O site do Ubuntu Linux, por exemplo, oferece links para processadores x86 (32 bits), Mac (chips PowerPC) e 64-bit (AMD64 ou EM64T).

Você pode ter se perguntado se é possível utilizar um sistema operacional de 32 bits com um processador de 64 bits e migrar o primeiro para uma versão adequada futuramente. Depende. O processador Intel Itanium é apelidado por alguns de "puro sangue", já que só executa aplicações de 64 bits. Assim, uma versão de 32 bits de um sistema operacional não roda nele. Por outro lado, processadores Athlon 64 são capazes de trabalhar tanto com aplicações de 32 bits quanto de 64 bits, o que o torna interessante para quem pretende usar um SO de 32 bits inicialmente e uma versão de 64 bits no futuro.

AMD64 e EM64T

Ao ser citadas anteriormente, o que significa as siglas AMD64 e EM64T:

- AMD64: originalmente chamado de x86-64, AMD64 (ou AMD64 ISA - Instruction Set Architecture) é o nome da tecnologia de 64 bits desenvolvida pela AMD. Um de seus destaques é o suporte às instruções de 32 bits (Legacy Mode);
- EM64T: sigla para Extended Memory 64-bit Technology, o EM64T é tido como a interpretação do AMD64 feita pela Intel. Devido a isso, recebeu de alguns a denominação iADMD64 (o "i" faz referência à primeira letra do nome da Intel).

Memória cache

Os processadores passam por aperfeiçoamentos constantes, o que os tornam cada vez mais rápidos e eficientes. No entanto, o mesmo não se pode dizer das tecnologias de memória RAM. Embora estas também passem por constantes melhorias, não conseguem acompanhar os processadores em termos de velocidade. Assim sendo, de nada adianta ter um processador rápido se este tem o seu desempenho comprometido por causa da "lentidão" da memória.

Uma solução para esse problema seria equipar os computadores com um tipo de memória muito mais rápida, a SRAM (*Static RAM*). Estas se diferenciam das memórias convencionais DRAM (*Dynamic RAM*) por serem muito rápidas, por outro lado, são muito mais caras e não contam com o mesmo nível de miniaturização, sendo, portanto, inviáveis. Apesar disso, a idéia não foi totalmente descartada, pois foi adaptada para **memória cache**.

A memória cache consiste em uma pequena quantidade de memória SRAM embutida no processador. Quando este precisa ler dados na memória RAM, um circuito especial chamado "controlador de cache" transfere blocos de dados muito utilizados da RAM para a memória cache. Assim, no próximo acesso do processador, este consultará a memória cache, que é bem mais rápida, permitindo o processamento de dados de maneira mais eficiente. Se o dado estiver no cache, o processador a utiliza, do contrário, irá buscá-lo na memória RAM, etapa essa que é mais lenta. Dessa forma, a memória cache atua como um intermediário, isto é, faz com que o processador nem sempre necessite chegar à memória RAM para acessar os dados dos quais necessita. O trabalho da memória cache é tão importante que, sem ela, o desempenho de um processador pode ser seriamente comprometido.

Os processadores trabalham, basicamente, com dois tipos de cache: cache L1 (Level 1 - Nível 1) e cache L2 (Level 2 - Nível 2). Este último é ligeiramente maior em termos de capacidade e passou a ser utilizado quando o cache L1 se mostrou insuficiente. Antigamente, um tipo distinguia do outro pelo fato da memória cache L1 estar localizada junto ao núcleo do processador, enquanto que a cache L2 ficava localizada na placa-mãe. Atualmente, ambos os tipos ficam localizados dentro do chip do processador, sendo que, em muitos casos, a cache L1 é dividida em duas partes: "L1 para dados" e "L1 para instruções".

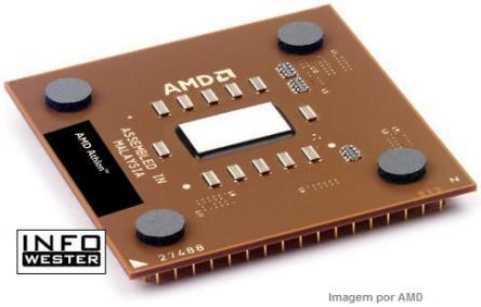


Imagem por AMD

Vale ressaltar que, dependendo da arquitetura do processador, é possível o surgimento de modelos que tenham um terceiro nível de cache (L3). Mas, isso não é novidade: a AMD chegou a ter um processador em 1999 chamado K6-III que contava com cache L1 e L2 internamente, algo incomum à época, já que naquele tempo o cache L2 se localizava na placa-mãe. Com isso, esta última acabou assumindo o papel de cache L3.

A foto acima mostra um processador AMD Athlon, com 64 KB de cache L1 para instruções, 64 KB de cache L1 para dados e 512 KB de cache L2.

Processadores com dois ou mais núcleos

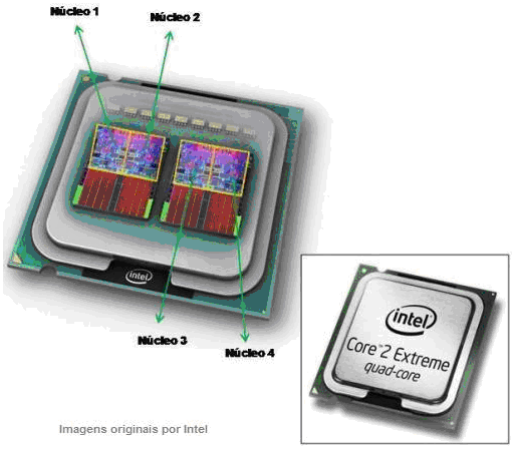
Há tempos que é possível encontrar no mercado placas-mãe que contam com dois ou mais slots para processadores. A maioria esmagadora dessas placas são usadas em computadores especiais, como servidores e *workstations*, os quais são utilizados em aplicações que exigem grandes recursos de processamento.

Até um passado não muito distante, o usuário tinha noção do quão rápido eram os processadores de acordo com a taxa de seu clock interno. O problema é que, quando um determinado valor de clock é alcançado, torna-se mais difícil desenvolver outro chip com clock maior. Limitações físicas e tecnológicas são os motivos para isso. Uma delas é a questão da temperatura: quanto mais megahertz um processador tiver, mais calor ele gerará.

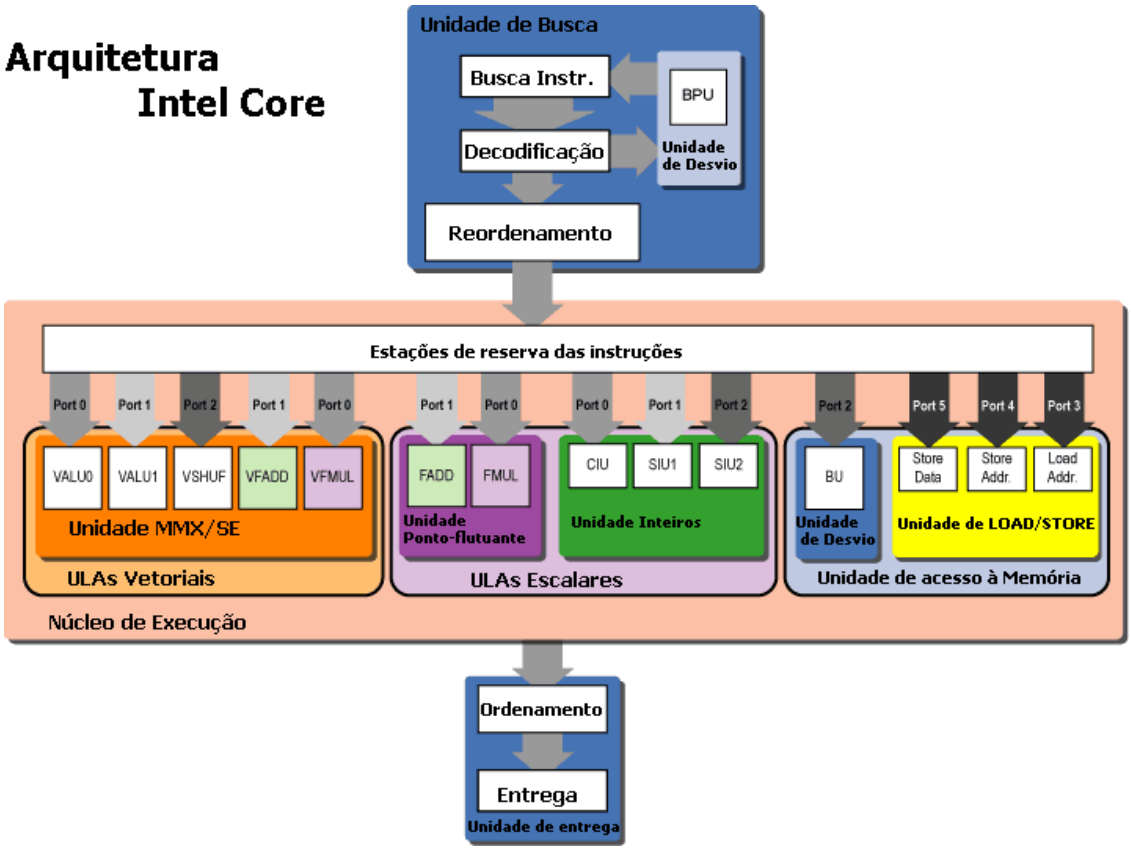
Uma das formas encontradas pelos fabricantes para lidar com essa limitação é fabricar e disponibilizar processadores com dois núcleos (*dual-core*) ou mais (*multi-core*). Mas, o que

isso significa? Processadores desse tipo contam com dois ou mais núcleos distintos no mesmo circuito integrado, como se houvesse dois processadores dentro de um.

Dessa forma, o processador pode lidar com 2 processos por vez, um para cada núcleo, melhorando o desempenho do computador como um todo. Em um chip de único núcleo, o usuário tem a impressão de que vários processos são executados simultaneamente, já que a máquina está quase sempre executando mais de uma aplicação ao mesmo tempo. Na verdade, o que acontece é que o processador dedica determinados intervalos de tempo a cada processo e isso ocorre de maneira tão rápida, que se tem a impressão de processamento simultâneo.



Na imagem acima, uma montagem que ilustra o interior de um processador Intel Core 2 Extreme Quad-Core (com 4 núcleos). Pelo menos teoricamente, é possível fabricar processadores com dezenas de núcleos. É importante ressaltar que ter processadores com dois ou mais núcleos não implica, necessariamente, em computadores que são proporcionalmente mais rápidos. Abaixo é exibido um diagrama simplificado de um núcleo de processamento (core) baseado na arquitetura Intel Core:



Processadores: i3, i5 e i7

O Intel Core i3

O Intel Core i3 é a linha de CPUs voltada aos menos exigentes. Por pertencer à nova linha Core, o i3 traz dois núcleos de processamento, tecnologia Intel Hyper-Threading (que possibilita a realização de mais tarefas), memória cache de 4 MB compartilhada (nível L3), suporte para memória RAM DDR3 de até 1333 MHz e muito mais.

As CPUs da linha Core i3 parecem fracos, contudo eles vieram para substituir a antiga linha Core2Duo. Qualquer Core i3 vem equipado com um controlador de memória DDR interno (o que já ocorre há muito tempo nos processadores da AMD), um controlador de vídeo integrado, Intel HD Graphics que opera na frequência de 733 MHz, e suporte para duplo canal para memória RAM (o que significa que as memórias trabalham aos pares).

Tecnologia Intel Hyper-Threading

As CPUs da linha Core i3 possuem apenas dois núcleos, pode imaginar que eles não durem muito mais. Contudo, com a utilização da Intel Hyper-Threading, os processadores i3 "ganham" dois núcleos a mais.

Por exemplo, ao se colocar um Core i3 ao lado de um Intel QuadCore, não há dúvidas de que o QuadCore terá um desempenho muito maior (em qualquer atividade). Claro que isso não significa que a nova tecnologia não serve para nada, muito pelo contrário. A Intel Hyper-Threading é ideal para momentos que se requer efetuar várias atividades simultaneamente. Essa tecnologia serve para que um núcleo consiga realizar duas atividades ao mesmo tempo, daí o motivo pelo qual a tecnologia, supostamente, faz os núcleos dobrarem em quantidade.

Os modelos da linha Intel Core i3 utilizam um soquete (encaixe na placa mãe), fator que forçou as montadoras a criarem placas exclusivas para eles. Conhecido como socket LGA 1156, esse novo tipo de soquete será utilizado para os processadores Intel Core i3, i5 e i7

O Intel Core i5 é o intermediário

O Intel Core i5 é encarregado de suprir as necessidades de porte intermediário, ou seja, aqueles mais exigentes que realizam tarefas mais pesadas. Disponível em modelos de dois ou quatro núcleos, os CPUs da linha i5 possuem até 8 MB de memória cache (nível L3) compartilhada, também utilizam o soquete LGA1156, controlador de memória DDR integrado, tecnologia Intel Hyper-Threading, tecnologia Turbo Boost e muito mais.

O que é e para que serve a tecnologia Turbo Boost?

A tecnologia Turbo Boost da Intel promete aumentar a velocidade do processador automaticamente. Segundo o site da Intel, esta tecnologia é inteligente e trabalha 100% do tempo verificando frequência, voltagem e temperatura do processador. Ao notar uma baixa em

um dos valores-padrão utilizados pelo CPU, este novo recurso aumenta a frequência e consegue um desempenho muito maior em qualquer aplicação.

Imagine que a temperatura do processador está abaixo do esperado e se deseja aumentar a velocidade. Com a utilização da tecnologia Turbo Boost o Intel Core i5 vai alterar a frequência ou a voltagem do CPU e logo será obtido um aumento significativo em desempenho. Especificamente dos modelos i5, há a possibilidade de um aumento de até 800 MHz no clock

O mais alto desempenho: Intel Core i7

A tecnologia de processamento é o i7 é voltada ao público entusiasta e profissional traz muitos benefícios e especificações de cair o queixo. Todos os CPUs da série Core i7 possuem quatro núcleos (o i7-980X com 6 núcleos), memória cache L3 de 8 MB, controlador de memória integrado, tecnologia Intel Turbo Boost, tecnologia Intel Hyper-Threading, tecnologia Intel HD Boost e o recurso Intel QPI.

Intel HD Boost? Para que serve?

Com o avanço constante dos processadores, os softwares foram forçados a evoluir. Existem softwares que trabalham com conjuntos de instruções específicas, as quais precisam estar presentes nos processadores para que o programa seja executado com a máxima performance. Os conjuntos de instruções principais são denominados como SSE, sendo que existem programas que utilizam instruções diferentes.

A linha de processadores Intel Core i7 trabalha com a tecnologia Intel HD Boost, a qual é responsável pela compatibilidade entre CPU e programas que usam os conjuntos de instruções SSE4. Tal característica possibilita um maior desempenho em aplicativos mais robustos que necessitam de um poder de processamento de alto nível.

Intel QPI

O recurso Intel QPI, ou *QuickPath Interconnect* (Interconexão de caminho rápido), serve para aumentar o desempenho do processador. Ao invés de aumentar a frequência ou a tensão, o recurso Intel QPI aumenta a largura de banda (o que permite a transmissão de mais dados) e diminui as latências. Vale salientar que este recurso só está presente nos CPUs Intel Core i7 da série 900 e possibilita taxas de transferência de até 25.6 GB/s.

Processador Intel i5

Tabela de Processadores– INTEL i3, i5 e i7.

Intel Core i7	**4ª Geração**	**3ª Geração**	**2ª Geração**	**1ª Geração**
Nome	i7-4770TE	i7-3770T	i7-2700K	i7-990X
Socket	LGA1150	LGA1155	LGA1155	LGA1366
Núcleos/Threads	4/8	4/8	4/8	6/12
Clock	2,3 GHz	2,5 GHz	3,5 GHz	3,46 GHz
Máx Clock	3,3 GHz	3,7 GHz	3,9 GHz	3,73 GHz
Cache L1	128 KB (programa) + 128 KB (dados)	128 KB (programa) + 128 KB (dados)	128 KB (programa) + 128 KB (dados)	192 KB (programa) + 192 KB (dados)
Cache L2	4 x 256 KB	4 x 256 KB	4 x 256 KB	6 x 256 KB
Cache L3	8 MB	8 MB	8 MB	12 MB
TDP	45 W	45 W	95 W	130 W
Gráficos	Intel® HD Graphics 4600	Intel® HD Graphics 4000	Intel® HD Graphics 3000	-
Memória	32 GB (DDR3-1600)	32 GB (DDR3-1600)	364 GB (DDR3-1333)	24 GB (DDR3-1066)

Intel Core i5	**4ª Geração**	**3ª Geração**	**2ª Geração**	**1ª Geração**
Nome	i5-4670T	i5-3570T	i5-2500T	i5-760T
Socket	LGA1150	LGA1155	LGA1155	LGA1156
Núcleos/Threads	4/4	4/4	4/4	4/4
Clock	2,3 GHz	2,3 GHz	2,3 GHz	2,80 GHz
Máx Clock	3,3 GHz	3,3 GHz	3,3 GHz	3,33 GHz
Cache L1	128 KB (programa) + 128 KB (dados)	128 KB (programa) + 128 KB (dados)	128 KB (programa) + 128 KB (dados)	128 KB (programa) + 128 KB (dados)
Cache L2	4 x 256 KB	4 x 256 KB	4 x 256 KB	4 x 256 KB
Cache L3	6 MB	6 MB	6 MB	8 MB
TDP	45 W	45 W	45 W	95 W
Gráficos	Intel® HD Graphics 4600	Intel® HD Graphics 2500	Intel® HD Graphics 2000	-
Memória	32 GB (DDR3-1600)	32 GB (DDR3-1600)	32 GB (DDR3-1600)	16 GB (DDR3-1333)

Intel Core i3	**4ª Geração**	**3ª Geração**	**2ª Geração**	**1ª Geração**
Nome	i3-4158U	i3-3250T	i3-2130	i3-560T
Socket	BGA1168	LGA1155	LGA1155	LGA1156
Núcleos/Threads	2/4	2/4	2/4	2/4
Clock	2,0 GHz	3,0 GHz	3,4 GHz	3,33 GHz
Máx Clock	-	-	-	-
Cache L1	64 KB (programa) + 64 KB (dados)	64 KB (programa) + 64 KB (dados)	64 KB (programa) + 64 KB (dados)	64 KB (programa) + 64 KB (dados)
Cache L2	2 x 256 KB	2 x 256 KB	2 x 256 KB	4 x 256 KB
Cache L3	3 MB	3 MB	3 MB	4 MB
TDP	28 W	35 W	65 W	73 W
Gráficos	Intel® Iris™ Graphics 5100	Intel® HD Graphics 2500	Intel® HD Graphics 2000	Intel® HD Graphics
Memória	16 GB (DDR3-1600)	32 GB (DDR3-1600)	32 GB (DDR3-1600)	16 GB (DDR3-1333)

Fonte: http://www.intel.com.br/content/www/br/pt/