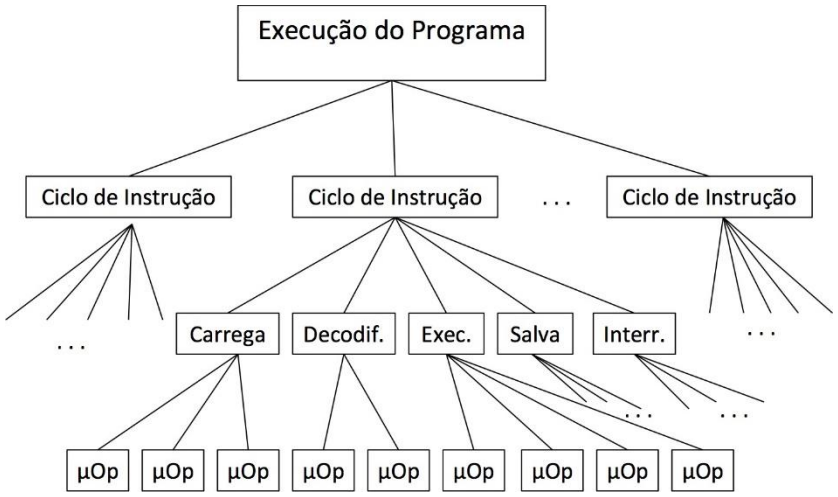
Unidade de Controle

A Unidade de Controle é a unidade interna à CPU responsável pelo controle da execução das instruções. Como a CPU é uma máquina de executar instruções, a Unidade de Controle é quem controla o principal funcionamento do computador.

O projeto de uma Unidade de Controle varia de arquitetura para arquitetura, mas há alguns pontos que são comuns a todas elas. Toda Unidade de Controle trabalha com a execução de Micro-operações. Como pode ser visto na figura abaixo, um programa é sempre executado instrução por instrução. Essas seriam instruções de máquina, compiladas a partir de Assembly, ou antes, a partir de uma linguagem de alto nível e depois compiladas para Assembly.

Cada instrução é executada através de um Ciclo de Instrução, como visto no capítulo anterior. Nessa ilustração o ciclo de instrução foi apresentado em cinco estágios: **Carrega Instrução, Decodifica, Executa, Salva Resultados e Checa Interrupção**. Mas ele pode variar de acordo com a arquitetura da Unidade de Controle e também de acordo com o tipo de instrução. Uma instrução com vários parâmetros que estão na memória, por exemplo, pode necessitar de um estágio a mais antes da execução para buscar os dados na memória. Já outra que utiliza apenas dados de registradores pode omitir esse passo e executá-lo no próprio estágio de execução.



Execução em Microoperações

Micro-operações

Na figura acima cada estágio do Ciclo de Instrução é quebrado em subestágios, Micro-operações. A Micro-operação é a operação atômica realizada pela Unidade de Controle para a execução de cada estágio. Cada vez que um estágio do Ciclo de Instrução for executado, as devidas Micro-operações são executadas pela Unidade de Controle, UC.

Essa organização é necessária para melhorar a organização da CPU e facilitar o projeto das UCs. Além disso, nos projetos modernos, é possível reutilizar Micro-operações de um estágio em outro. Por exemplo, para executar uma instrução, é necessário executar uma Micro-operação para buscar um dado que está em um registrador (ou memória) para a ULA. Se houver vários dados envolvidos, a única tarefa a ser realizada é pedir para a UC executar a mesma Micro-operação de buscar dado, mas agora com endereço diferente.

Dessa maneira, todas as instruções que chegam à UC são quebradas em estágios (lembre-se do Pipeline), que por sua vez, separados em sequências de Micro-operações e, só então, são executadas. Agora vamos apresentar alguns exemplos de como os principais estágios de execução são organizados em Micro-operações.

Busca de Instruções

Neste estágio uma nova instrução deve ser buscada da memória e trazida para a CPU para que possa ser decodificada e executada em seguida. O endereço da instrução a ser buscada está sempre no Contador de Programa (PC) e a instrução buscada da memória é armazenada no Registrador de Instrução (IR). A seguir é apresentado as Micro-operações para realizar a Busca de Instrução.

A seguir cada Micro-operação é apresentada ao lado de uma determinação do tempo em que ela será realizada. Nesse caso, 3 unidades de tempo são necessárias (**t1, t2** e **t3**). Imagine que cada unidade de tempo é 1 ciclo de clock (as micro-operações mais complexas levam mais tempo).

t1: **MAR <- (PC)**
Memória <- fetch;
t2: **MBR <- (Memória)**
PC <- (PC) + 1
t3: **IR <- (MBR)**

No tempo t1 o endereço guardado no registrador PC é copiado para o registrador MAR, que é o registrador conectado ao Barramento de Endereço. Ao mesmo tempo, a CPU envia para a memória um sinal de *fetch*, indicando que precisa ser trazida uma nova instrução. No tempo t2 a memória lê o endereço contigo no Barramento de Endereço (que é sempre o mesmo de MAR), busca a instrução e a escreve no Barramento de Dados. Como o MBR sempre reflete aquilo que está no Barramento de Dados, o MBR agora contém a instrução trazida da memória. Esta tarefa é realizada essencialmente pela memória, a ULA está livre para executar outra Micro-operação, e aproveita para adicionar 1 ao endereço do PC. Esta operação de incremento vai garantir que a próxima instrução a ser buscada será a do endereço seguinte da memória. Finalmente, no tempo t3, a instrução que foi trazida da memória e salva em MBR pode ser agora salva em IR.

A instrução precisa sempre estar em IR porque na faze de decodificarão, a Unidade de Controle vai buscar lá que instrução deve executar.

A mesma Micro-operação pode ser executada de forma diferente, como mostrada a seguir.

t1: **MAR <- (PC)**
Memória <- fetch;
t2: **MBR <- (Memória)**
t3: **PC <- (PC) + 1**
IR <- (MBR)

Nesta segunda opção, o incremento de PC foi passado para o tempo **t3**, ao invés de **t2**, o que gera o mesmo resultado.

A quantidade de Micro-operações que podem ser executadas no mesmo ciclo de clock depende da arquitetura do processador. Por exemplo, se há apenas uma ULA, então só é possível executar uma única operação lógica, ou aritmética a cada ciclo de clock. O mesmo serve para o uso acesso à memória com MAR e MBR. Se houver um barramento exclusivo para instruções, separado do barramento de dados e uma memória de instrução separada da memória de dados (quase todos processadores hoje possuem), e então é possível buscar uma instrução no mesmo ciclo de clock que se busca um dado.

Busca Indireta

Outro tipo de Micro-operação muito utilizado é a Busca Indireta. Ela trata de buscar um dado que está na memória e trazer para a CPU para ser utilizado pela instrução. O termo “indireta” indica que o dado não está diretamente na CPU (registrador), mas na memória.

Imagine que a instrução em questão seja a soma de 2 números A e B, ambos na memória. Esta instrução foi buscada no estágio anterior, portanto no ciclo seguinte ela estará no registrador IR. Então, os endereços de A e B estão presentes na instrução que agora está em IR. A Busca Indireta deve então ser realizada para A e depois para B, da seguinte forma:

t1: **MAR <- (IRendereco)**
Memória <- read
t2: **MBR <- (Memória)**
t3: **ACC <- (MBR)**

No instante **t1** o endereço do dado contido em IR é passado para o registrador de endereço MAR. Ao mesmo tempo a CPU envia para a memória um sinal de leitura (read), avisando que deve ser feita uma busca indireta. No instante seguinte, **t2**, o conteúdo do dado é trazido da memória para o MBR através do Barramento de Dados, e no último passo, em **t3**, o conteúdo agora em MBR é levado para um registrador para que seja utilizado na operação, geralmente o Acumulador (ACC).

Execução

Após a busca da instrução e dos dados necessários, é hora de executar a instrução na Micro-operação de execução. Mantendo o exemplo da soma A e B apresentado anteriormente, a Unidade de Controle terá que fazer a Busca Indireta de A e B para depois realizar a soma. Supondo que A seja salvo em ACC, ele deve ser transferido para outro registrador, digamos R1 antes de Busca Indireta por B. Assim, a execução seria:

t1: R1 <- (ACC)
// Busca indireta por B
t2: MAR <- (IRendereco)
Memória <- read;
t3: MBR <- (Memória)
t4: ACC <- MBR
t5: ACC = R1 + ACC

Em t1 o conteúdo de A salvo em ACC será transferido para o registrador R1. Nos intervalos de t2 até t4 seria feita a Busca Indireta por B. E, finalmente, no instante t5 a soma de R1 e ACC seria realizada e salva no acumulador ACC.

Salvar Resultado

Após esse último passo, o conteúdo de ACC com o resultado da operação deve ser transferido para o local de destino. Isso é feito no estágio de Salvar Resultado. Se o resultado for salvo num registrador, a operação é direta e feita num único ciclo de clock. Mas se precisar salvar o resultado na memória, uma escrita indireta deverá ser realizada para salvar o conteúdo de ACC na memória.

t1: MAR <- (IRendereco)
t2: MBR <- (ACC)
Memória <- write

Para tal, inicialmente em t1 o endereço da variável de memória que precisa ser salva é passado para o MAR. O conteúdo de ACC é passado para o MBR no ciclo seguinte (t2), ao mesmo tempo em que a CPU envia para a memória um sinal de escrita. Ao receber esse sinal, a memória traz o conteúdo de MBR e o salva no endereço representado por MAR.

Salto Condicional

Uma instrução muito comum executada pelos computadores são os saltos condicionais. Ela indica que se uma determinada condição for satisfeita, a execução não deve ser a da instrução seguinte, mas a indicada pela instrução. Imagine uma instrução “Salta se zero”, com dois parâmetros, X e Y. O X seria a variável a ser testada e o Y o endereço para onde a execução deve saltar caso o valor de X seja 0.

Desta forma, as micro-operações seriam as seguintes:

// Busca indireta por X:
t1: MAR <- (IRenderecoX)
Memória <- read;
t2: MBR <- (Memória)
t3: ACC <- MBR
t4: se ACC == 0, PC = (IRenderecoY)

Inicialmente, de t1 a t3, seria buscado o conteúdo de X na memória. No último ciclo t4, o conteúdo de ACC seria comparado com 0 e se forem iguais, o conteúdo de PC será o endereço da variável Y, também presente em IR. Observe que, caso contrário, nada precisa ser feito, o PC continuará como antes e a próxima instrução depois da atual será executada.

Tipos de Micro-operações

Como já foi possível observar através dos exemplos apresentados, há quatro tipos básicos de Micro-operações de uma Unidade de Controle. São eles:

- Transferência de dados entre registradores
- Transferência de dados de registrador para o exterior da CPU
- Transferência de dados do exterior da CPU para um registrador
- Operação lógica e aritmética

A transferência de dados de um registrador para outro é a mais simples das Micro-operações e geralmente é feita num único ciclo de clock. Já a movimentação de dados de ou para o exterior da CPU pode ser mais complexa. Para facilitar muitos computadores mapeiam todos dispositivos de Entrada e Saída com se fossem memória. Ou seja, para a CPU, enviar um dado para um dispositivo seria como escrever um dado na memória, bastando usar um endereço diferente. Isso facilita bastante a operação da Unidade de Controle, mas pode limitar a quantidade de endereços de memória disponíveis para os programas. As operações de transferência de dados são complexas também porque levam um tempo não conhecido para serem executadas. Se o dado estiver na memória Cache o acesso é mais rápido, se estiver na Memória Principal levará mais tempo, e se tiver num dispositivo externo, um Disco Rígido, por exemplo, pode levar ainda mais.

As operações lógicas e aritméticas podem ser mais rápidas ou mais lentas dependendo de cada uma delas. Operações com números de Ponto Flutuante tendem a levar mais tempo do que aquelas com números inteiros. As operações trigonométricas são as mais lentas que o computador pode operar.

Decodificação

A execução das Micro-operações é sempre ordenada pela Unidade de Controle. Isso é feito no estágio de Decodificação, a partir da leitura da instrução presente em IR. O primeiro da decodificarão é ler o código da instrução para conhecer seu tipo. Dependendo do tipo, a instrução é quebrada numa quantidade específica de Estágios e cada estágio no seu respectivo grupo de Micro-operações. Cada vez uma que Micro-operação se encerra, a Unidade de Controle checa qual será a próxima e envia os sinais para os devidos registradores, para ULA e dispositivos envolvidos, como memória ou dispositivos de Entrada e Saída.

Dessa forma, podemos dizer que a Unidade de Controle possui duas funções principais, a execução e o sequenciamento das instruções. Nessa última função a Unidade de Controle deve saber o exato momento que uma Micro-operação concluiu para executar a próxima, e quando a última Micro-operação for executada, iniciar um novo Estágio, e quando o último Estágio for concluído, executar a próxima instrução do programa.