Unidade 10: A Unidade Lógica Aritmética
e as Instruções em Linguagem de Máquina

Prof. Daniel Caetano

Objetivo: Apresentar as funções o mecanismo de atuação da Unidade Lógica Aritmética e iniciar a descrição dos tipos de instrução que o processador executa.

Bibliografia:

- STALLINGS, W. **Arquitetura e organização de computadores**. 5ed. São Paulo: Ed. Pearson Prentice Hall, 2003.
- MURDOCCA, M. J; HEURING, V.P. **Introdução à arquitetura de computadores**. S.I.: Ed. Campus, 2000.

INTRODUÇÃO

* CPU: Duas partes principais
  - ULA + UC
* Iniciar pela ULA: execução dos cálculos

Nesta aula iniciaremos o estudo do funcionamento e operação de uma Unidade Central de Processamento (CPU), através do estudo de cada um de seus elementos. O primeiro elemento que será estudado é a Unidade Lógica Aritmética; não porque ele é o elemento mais simples ou porque ele é o primeiro elemento na longa seqüência de operações que uma CPU executa, mas porque ele é o elemento principal de uma CPU: é a Unidade Lógica Aritmética (ULA) quem executa a maior parte das instruções de uma CPU.

Como veremos posteriormente, praticamente todas as outras partes da CPU são voltadas a uma única finalidade: trazer instruções para que a ULA as processe, bem como armazenar os dados resultantes.

1. O PROCESSADOR E OS REGISTRADORES

É possível pensar no processador como um conjunto de funcionários praticamente sem memória alguma. É como se dois funcionários, chamados ULA e Controle, estivessem traduzindo um texto em um determinado andar da empresa, mas o dicionário, chamado Memória, ficasse apenas em outro andar da empresa. Isso significa que, para cada palavra que ULA fosse traduzir, o Controle teria que ir até o outro andar, procurar a palavra na Memória e só então voltar para traduzir.

Isso tem dois problemas: 1) é extremamente lento; 2) certamente o Controle já teria esquecido a informação que foi buscar quando chegasse de volta à sua mesa de trabalho.

A parte da lentidão foi resolvida com a contratação de um funcionário subalterno, chamado Cache. O Cache fica no elevador e, quando ULA precisa de um dado, ele pede ao Controle, que grita para o Cache, que vai buscar o dado, volta e grita de volta para o Controle a resposta. Isso ainda é um pouco lento, porque o Cache às vezes ainda tem que subir e descer o elevador, mas de alguma forma às vezes ele não precisa, porque ele adivinha o que o Controle vai querer saber e se informa antes! Em algumas situações, a resposta dele é imediata!

Bem, ocorre que mesmo com a contratação deste novo funcionário (o Cache), o Controle demora um pouco a responder às dúvidas do ULA que, neste meio tempo, se esquece do que estava traduzindo e, quando ele se lembra, tanto ele quanto o Controle já esqueceram a informação que o Cache havia acabado de passar.

Tomando conhecimento deste problema, a diretoria instituiu que os funcionários Controle e ULA passassem a adotar "pequenos papéis", verdadeiras "colas", para anotar uma informação assim que o Cache a informa; desta maneira, não havia mais erro: quando ULA se lembra o que ia fazer com a informação, ele olha a "cola" e a informação está lá, para que ele possa trabalhar com ela.

Como a empresa tem certificação de qualidade total ISO 9001, a diretoria achou por bem que estas colas fossem em uma quantidade pequena e que fossem reaproveitáveis, para contribuir com a natureza e não desperdiçar recursos. Para facilitar a identificação das mesmas (e para que ninguém de outro departamento levasse as "colas" embora), a diretoria indicou dois dizeres nos mesmos: a palavra "Registrador", pois é um local onde informações devem ser registradas e o nome deste registrador, que normalmente é alguma letra: A, B, C...

Os funcionários Controle e ULA decidiram que, como não há muitos *registradores*, para evitar confusão, algumas tarefas específicas usariam sempre os mesmos *registradores*. Definiram ainda que o Controle pode escrever e ler em qualquer *registrador*, mas a ULA pode escrever apenas em alguns (em especial no A), embora possa ler praticamente todos. Tanto o Controle quando a ULA perceberam ainda que havia informações que não caberiam em apenas um *registrador*. Por esta razão, eles combinaram que, nestes casos, seriam usados pares de *registradores* para registrar a informação completa.

A idéia funcionou tão bem que todas as empresas concorrentes copiaram, embora muitas vezes deem nomes diferentes para os *registradores*.

1.1. O Funcionamento Real

A analogia ajuda a entender como o computador funciona, mas ela não é exatamente precisa. Na prática, a Unidade de Controle é quem comanda as operações e a ULA apenas responde às requisições do Controle. Ocorre que a ULA não tem ligações com a memória

física do computador (o acesso a elas é bastante complexo); mas então, como ela recebe dados?

Bem, para que a ULA possa trabalhar, um processador tem uma pequena quantidade de memória (muito rápida) dentro de seu encapsulamento. Cada posição de memória do processador é chamada de *registrador* e é ligada diretamente à ULA e ao Controle.

Assim, quando o Controle pretende enviar uma ordem à ULA (ordem esta chamada *instrução*), ele precisa preencher os *registradores* previamente, com as informações que a ULA irá precisar para executar a operação. Observe a Figura 1 para compreender melhor o processo:

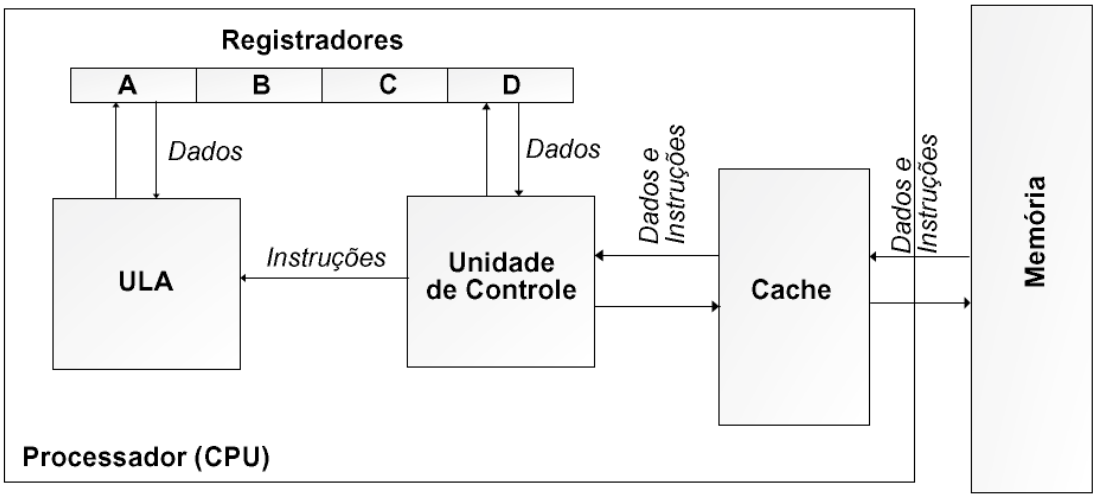

Figura 1: Interligações da ULA com Unidade de Controle e Registradores

Desta maneira, quando a Unidade de Conotrole envia uma instrução do tipo "ADD A,B" (adicionar B em A) para a ULA, esta última irá ler os registradores A e B, somá-los e colocar o resultado em A.

2. OPERAÇÕES EXECUTADAS PELA ULA

Como foi apresentado na seção anterior, a ULA responde à requisições da Unidade de Controle que, como veremos nas aulas seguintes, é responsável por preparar os registradores e entregar instruções decodificadas para a ULA.

A ULA só realizará, então, operações simples de aritmética e lógica, estando os parâmetros das mesmas em um ou mais registradores (mas nunca diretamente na memória). São operações como *adição* (ADD), *subtração* (SUB), *multiplicação* (MUL), *divisão* (DIV), *e* (AND), *ou* (OR), *ou exclusivo* (XOR), *não* (NOT) dentre outras mais específicas.

Os registradores mais importantes acessados pela ULA são o *Acumulador*, freqüentemente chamado de "A" (ou AX ou EAX na arquitetura x86). Outro registrador

importante, existente em algumas arquiteturas, é o *Flags*, normalmente chamado de "F" (ou FLAGS ou EFLAGS ou RFLAGS na arquitetura x86). Nas arquiteturas RISC, entretanto, os nomes de registradores são pouco significativos, normalmente variando de R1 a Rxx (a arquitetura RISC será melhor detalhada no futuro).

Note que instruções de leitura e escrita em memória (LOAD, MOVE, STORE etc) não são processadas pela ULA. Algumas instruções como saltos relativos à posição atual usam a ULA para o cálculo da nova posição de memória, mas grande parte do trabalho destas instruções (assim como no caso de instruções que fazem operações com dados na memória principal) é feito pela Unidade de Controle.

3. INSTRUÇÕES DA CPU

Na seção anterior falamos brevemente sobre as instruções da CPU. O que são essas instruções e quantas existem?

As "instruções" como comentadas anteriormente (ADD, LOAD, MOVE...) são, na verdade, "apelidos mnemônicos" dados a instruções binárias, para facilitar a sua memorização. Quando um computador lê uma instrução na memória, ele lê, na verdade, um conjunto de bits. Para entender melhor, vamos tomar um exemplo.

A instrução **ADD A,r** , do microprocessador Z80, serve para somar o valor de qualquer registrador **r** ao registrador A, isto é, realiza a seguinte tarefa: **A = A + r** . Pois bem, essa instrução, em bits, é descrita da seguinte forma:

1	0	0	0	0			
⟵ ADD A,r ⟶					⟵ r ⟶		

Dependendo dos valores dos bits "r", a operação executada usará um registrador diferente. Os valores que esses bits podem assumir são:

Registrador	**A**	**B**	**C**	**D**	**E**	**H**	**L**
Bits	111	000	001	010	011	100	101

Assim, a instrução ADD A,D - ou seja, A = A+D - pde ser representada assim:

1	0	0	0	0	0	1	0
⟵ ADD A,r ⟶					⟵ r ⟶		

Note que cada instrução tem uma "sintaxe de bits" diferente: em cada uma delas, os bits relativos à instrução e à designação dos operandos é diferente. Vejamos, como exemplo, uma outra instrução: **INC r** .

A instrução INC r **incrementa** o valor do registrador **r**, isto é, realiza uma operação que pode ser representada como **r = r + 1** . Em binário, essa instrução é especificada da seguinte forma:

0	0				1	0	0
← INC r →		← r →			← INC r →		

O código para indicar o registrador é o mesmo usado na instrução ADD A,r.

Ainda que esse tipo de especificação permita uma certa diversidade de operações, este número é um tanto limitado. Considerando que usaríamos sempre 3 bits para indicar um registrador, com 8 bits teríamos apenas mais 5 para indicar a instrução, o que nos limitaria a, basicamente, 32 instruções. Sendo assim, algumas instruções possuem mais de um byte (instruções de 16 bits, 32 bits...).

3.1. Instruções com Dados

Algumas instruções, como as apresentadas anteriormente, possuem toda a informação necessária em si mesma. Mas esse não é sempre o caso. Imagine, por exemplo, a instrução

ADD A,n

Essa instrução adiciona um número **n** qualquer, de 8 bits, ao valor existente no registrador A. É o equivalente a fazer **A = A + n** . Nesse caso específico, é usada **uma instrução de 8 bits que exige um dado de 8 bits em sequência**. Ou seja: a instrução ADD A,n exige que, na posição de memória imediatamente seguinte, exista o número a ser adicionado ao registrador A. Na memória isso fica assim:

1	1	0	0	0	1	1	0
← ADD A,n →							

← n →							

Assim, para realizar uma operação **ADD A,0x24** , indicamos da seguinte forma:

1	1	0	0	0	1	1	0
← ADD A,n →							

0	0	1	0	0	1	0	0
← n →							

NOTA: Como vimos, a ULA só tem acesso aos registradores. Como é possível somar um número da memória diretamente a um registrador? **Como** isso ocorre será explicado na próxima aula.

3.2. Conjuntos de Instruções

Cada CPU tem seu próprio conjunto de instruções (tanto em binário quanto a forma "mnemônica" oficial) e é um tanto inútil ficar estudando diversas delas antes de ser necessário. Entretanto, os tipos de instruções comuns na grande maioria dos processadores pode ser dividido em algumas categorias:

a) Transferência de Dados (MOVE, STORE, LOAD, EXCHANGE, PUSH, POP...)
b) Operações de Entrada/Saída (READ, WRITE...)
c) Operações Aritméticas (ADD, SUB, MULT, DIV, INC, DEC...)
d) Operações Lógicas (AND, OR, NOT, XOR, TEST, COMPARE, SHIFT...)
e) Transferência de Controle (JUMP, CALL, HALT...)
f) Operações de Conversão (TRANSLATE, CONVERT...)

As operações (a) e (b) são executadas basicamente pela Unidade de Controle (UC). As operações (c) a (f) podem envolver apenas a ULA (se forem operações apenas com registradores) ou podem envolver a ULA e a UC (se os dados estiverem na memória).

4. MODOS DE ENDEREÇAMENTO

Até agora, vimos instruções que trabalham com dois tipos de endereçamento:

a) Endereçamento a Registradores: aquelas em que o valor já está em um registrador (nenhum acesso à memória é necessário).
b) Endereçamento Imediato: aquelas que o valor segue a instrução, na memória.

Como foi comentado, como a ULA só tem acesso aos registradores (endereçamento do tipo (a)), o endereçamento imediato só é possível com o auxílio da Unidade de Controle.

> NOTA: A Unidade de Controle (UC) será o foco da próxima aula!

Entretanto, usar imediatos para acessar informações é extremamente limitado. Sendo assim, a CPU usualmente fornece uma grande quantidade de maneiras de ler dados da memória.

Uma das instruções mais simples que usa todas as possíveis maneiras de acesso à memória - e por isso vamos usá-la como exemplo - é a que lê um dado da memória para um registrador.

Essa instrução é, nromalmente, chamada de LOAD. Na arquitetura do Z80, que estamos usando como exemplo, ela é especificada como LD (de Load). Vejamos as formas possíveis de acessar a memória nessa instrução.

c) Endereçamento Direto: o número da posição e memória a ser lida é representado como um dado imediato. Por exemplo:

LD A,(e)

Essa instrução procura pelo endereço **e** na memória e lê o seu conteúdo no registrador A. No Z80, essa é uma instrução que, incluindo o endereço, tem 24 bits: 8 bits para a instrução e 16 bits para o endereço a ser lido (já que os endereços do Z80 possuem 16 bits). Exemplo: **LD A,(0x7200**)

Essa instrução **não** irá carregar o valor 0x7200 no registrador A, mas sim carregar o registrador A com o valor armazenado na posição de memória 0x7200.

d) Endereçamento Indireto: o número da posição e memória a ser lida está armazenado em um registrador. Por exemplo:

LD A,(rr)

Essa instrução procura na memória pelo endereço indicado pelo registrador **rr** e, quando o encontra, armazena o valor da memória no registrador A. No Z80, essa é uma instrução que tem apenas 8 bits, já que o registrador que armazena a informação do endereço já está incluído na instrução. Exemplo: **LD A,(HL**).

Se HL armazenar o valor 0x2010, essa instrução **não** irá carregar o valor 0x2010 no registrador A, mas sim carregar o registrador A com o valor armazenado na posição de memória 0x2010.

NOTA: Algumas arquiteturas permitem acesso indireto com um valor imediato, isto é, seria o mesmo que realizar LD A,((e)), em que o endereço de memória e indicaria o endereço do qual se quer ler o dado. Esse tipo de endereçamento é relativamente incomum.

e) Endereçamento por Deslocamento: o número da posição e memória a ser lida está armazenado em um registrador. Por exemplo:

LD A, (r + n)

Essa instrução calcula o endereço de memória a ser lido, somando o valor do registrador **r** a valor **n**. O endereço resultante é usado para ler o valor da memória para o registrador A. Exemplo: **LD A,(IX + 1**).

Nesse caso, se IX armazenar o valor 0x1010, essa instrução **não** irá carregar o valor 0x1011 no registrador A, mas sim carregar o registrador A com o valor armazenado na posição de memória 0x1011.

NOTA: dependendo do "tamanho em bits" do registrador e do número, esse tipo de endereçamento ganha um nome diferente: se o **registrador contém um endereço** e o número representa apenas um deslocamento, esse endereçamento se chama **Endereçamento via Registrador Base**. Por outro lado, se o **número é um endereço** e o registrador contém o deslocamento, esse acesso é chamado de **Endereçamento por Indexação**.

f) Endereçamento por Pilha: o número da posição e memória a ser lida ou escrita está em um registrador especial, que é operado pela unidad de controle. Neste caso, a instrução LOAD não funciona, teremos de usar o exemplo das instruções PUSH e POP.

PUSH r

Essa instrução insere o valor armazenado no registrador **r** na pilha, que é uma fila do tipo LIFO (Last In Firt Out ou, em português, o último a entrar é o primeiro a sair). PUSH serve para armazenar valores, POP serve para recuperá-los.

5. BIBLIOGRAFIA

STALLINGS, W. **Arquitetura e organização de computadores**. 5ed. São Paulo: Ed. Pearson Prentice Hall, 2003.

MURDOCCA, M. J; HEURING, V.P. **Introdução à Arquitetura de Computadores**. S.I.: Ed. Campus, 2000.