Unidade: Unidade Lógica e Aritmética e Registradores

Unidade: Unidade Lógica e Aritmética e Registradores

UNIDADE LÓGICA E ARITMÉTICA E REGISTRADORES

O Processador é um chip com milhares de componentes encapsulados, as junções destes componentes formam algumas unidades básicas para que ele trabalhe de forma adequada. Nesta aula você irá estudar com maiores detalhes os principais componentes como registradores e unidade de controle, você ainda irá conhecer os mecanismos para que um dado ou instrução possa vir da memória principal para o processador para que seja processado.

1) Unidade lógica e Aritmética

A unidade lógica e aritmética (*aritmetic logic unit*) – ULA é um modulo capaz de realizar um conjunto de funções aritméticas e lógicas. Para atingir este objetivo, uma ULA tem vetores de entrada e saída de dados, bem como entradas de controle; estas entradas de controle são usadas para selecionar a função específica a ser executada em um tempo determinado. Entende-se por operação lógica comparações (de igualdade, maior que, menor que) e operações aritméticas como operações matemáticas (soma, subtração, divisão, exponenciação, divisão, atribuição de valores etc.). As operações são realizadas sucessivamente, observando os critérios de prioridade de operações lógicas (e, ou - maior que, menor que) e aritméticas (ordem de execução conforme o tipo de operação: multiplicação e divisão - soma e subtração etc.), e as 'informações' são gravadas temporariamente nos Registradores Internos do Processador. Cada registrador tem função específica e papel fundamental na realização das operações.

Alguns tipos de ULA são disponíveis na forma de circuitos integrados (chips), como o CI 74181.

2) Registradores de dados

Todos os processadores têm alguns registradores, eles estão lá para controlar a execução de programas e armazenar resultados temporários além de outras finalidades específicas.
Os registradores são locais de armazenamento interno ao processador, as instruções do processador operam diretamente sobre os valores que estão armazenados nos registradores. Todos os dados devem ser carregados para um registrador antes que seu conteúdo seja processado. Existem algumas operações que podem carregar os dados da memória principal para os registradores e depois enviar o resultado de volta diretamente para a memória principal. Cada registrador tem um tamanho que determina a quantidade máxima de dados que podem ser processados. Para você ter uma idéia os processadores Pentium tem registradores cujo tamanho é de 32 bits.

Portanto para que a CPU trabalhe de forma eficiente, tanto para interpretar um código de máquina, como para controlar a memória, ela usa o que chamamos de registradores. Você pode idealizá-los como sendo variáveis numéricas internas da CPU, ou seja, não pertencem a um endereço de memória é podem ser manipulados tanto pela máquina como pelo programa. Estes registradores possuem fins específicos que podem assumir valores inteiros de 0 a 65.536. São usados, entre outras funções, para mover conteúdo da memória, interpretar um código de máquina (programa), fazer operações aritméticas e operações de comparações.

Para que você tenha uma idéia o processador 8086 tem 14 registradores, vamos descrever cada um deles abaixo.

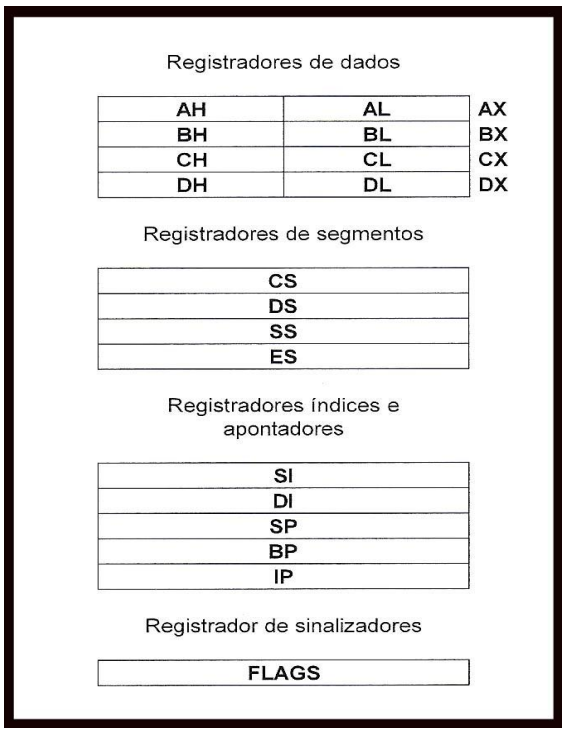

Figura 1: Registradores comuns

AX – Acumulador
BX – Base
CS – Contador
DX – Dados
CS – Segmento de código
DS – Segmento de dados
SS – Segmento de Pilha
ES – Segmento extra
IP – Contador de Programa
SP – Ponteiro de Pilha
BP – Base de Pilha
SI – indexador de Origem
DI – Indexador de destino
FLAGS – Estado

Os registradores CS, DS, SS, e ES são utilizados para apontar segmentos de dados com o código que se deseja trabalhar. O conteúdo do CS indica o segmento corrente do código do programa que está sendo executado. O DS indica o segmento de dados corrente, onde estamos nos referenciando. O SS é o utilizado para o segmento da pilha. O ES é usado como segmento extra, que permite acessar outros segmentos de dados, sem a necessidade de alteração do DS, SS ou CS.

O programa interpretado é controlado pelo IP, também conhecido como contador de programas. Ele aponta para a próxima instrução de máquina a ser executada no segmento definido por CS. Sempre que uma instrução é executada, o IP é incrementado pela CPU de acordo com o número de bytes da instrução executada.

Os registradores X (AX, BX, CX e DX) possuem uma característica diferente dos outros. Podemos trabalhar com eles com valores inteiros (16 bits).

No contexto de um programa de máquina o qual é composto por códigos considerados a baixo nível, usamos o registrador AX como valor provisório e para operações aritméticas, este registrador é também conhecido como acumulador. O registrador BX trabalha para endereçar memória funcionando como um endereço base e indexação. O registrador CX é usado como contador geralmente de Loops de controle. O registrador DX é aplicado, na maioria das vezes, para apontar memória de dados. Estas aplicações para os registradores X não são obrigatórias, mas o conjunto de comandos interpretados pela CPU parte dos princípios descritos acima.

Em quase todas as linguagens de programação, utilizamos as variáveis I e J para indexar um vetor de dados qualquer ou, para controlar um laço de repetição. As linguagens possuem registradores de indexação SI e DI que são utilizados de uma forma bem parecida às variáveis I e J em programas considerados de alto nível, por exemplo, a linguagem Java. O SI, na maioria das vezes, indexa na memória uma área de origem e o indexa na memória uma área DI de destino. Eles são muito úteis para copiar ou varrer a memória. Já o SP e o BP são usados pela pilha de controle, este conceito veremos mais adiante em uma próxima aula.

O último registrador a analisar tem uma importância fundamental, constitui-se no registrador chamado flags, ele nos permite determinar o resultado de uma comparação, além de outros controles disponíveis. Ele trabalha de forma diferenciada dos demais registradores, uma vez que não utilizamos seu valor (16 bits) mais usamos o resultado de seus bits de forma isolada.

Veja, na figura abaixo a interligação de todos estes registradores com outros componentes da CPU.

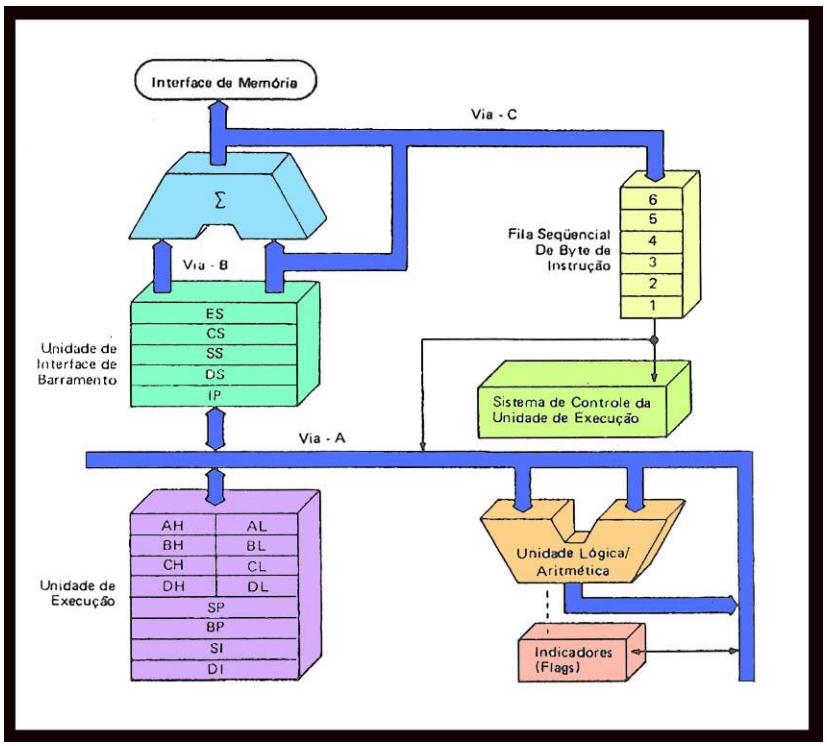

Figura 2: Disposição dos registradores no processador

Não se preocupe em decorar todas estas informações e sim compreender para que serve cada registrador. Lembrando que todos estes registradores são pequenos espaços de memórias de alta velocidade que são implementados dentro do microprocessador.

A Unidade Lógica e Aritmética tem uma função muito importante, pois é nela que são processados os dados vindos da memória principal. Além disto, os registradores têm papel fundamental na armazenagem temporária dos dados e instruções que trafegam pelo processador. Todos estes componentes são feitos de transistores e tem funções específicas de acordo com a disposição como são montados dentro do chip. As instruções de um processador são baseadas nestes componentes a fim de proporcionar um bom funcionamento do computador.

Para se programar em linguagem de baixo nível, alguns detalhes devem ser conhecidos, dentre estes detalhes estão conhecer a função de cada registrador e as instruções que os afetam. O registrador FLAG tem uma função importante, pois nele encontram-se bits que permitem verificar o estado de algumas operações ou comparações. Neste processo a ULA tem papel fundamental, pois é nela que alguns dos registradores estão implementados. Nesta aula você irá estudar o registrador FLAG e a ULA em maiores detalhes.

2.1) Registradores Especiais de estados

Há um registrador na CPU que é considerado muito importante no processamento dos dados, este registrador chama-se FLAGS e permite determinar o resultado de uma comparação, entre outros tipos de controles. Ele trabalha de forma diferenciada dos demais registradores vistos até aqui, uma vez que não utilizamos o seu valor numérico de 16 bits, mas sim o valor dos seus bits isoladamente, abaixo encontra-se a descrição destes bits.

5	4	3	2	1	0										

Onde:

O – Overflow (Estouro) – A operação aritmética estourou, ou seja, o valor do resultado é maior que um byte (255). Esse flag é usado quando o Carry Flag deixa a desejar em cálculos como:

AL = 254(sem sinal) ou –2(com sinal)

AL = 254 + 5 = 259

Nesse caso, o Carry Flag será configurado com o valor 1, pois o resultado superou o limite de um byte.

Assim, se o resultado estiver dentro dos limites do inteiro com sinal, o bit “O” será zero, caso contrário, se o valor do cálculo exceder esse limite, o bit “O” será um.

D – Direção – Se igual a 1 (um) indica decremento, se 0 (zero), indica o incremento dos indexadores para as operações de tratamento de memória.

I – Interrupção – Se estiver com o valor 1, habilita interrupções.

T – Trap – Utilizado para programas de depuração.

S – Sinal – Se em uma operação o resultado for um número negativo, então S será igual a 1.

Esse flag é usado para saber se um valor é positivo (S = 0) ou negativo (S = 1), se ele for declarado como um número com sinal.

Z – Zero – A operação resultou em um valor zero se Z =1.

- Se esta flag estiver com o valor 1, indica que o resultado de uma

operação sobre um registrador ou área de memória tem o valor zero, isto é útil em comparações.

Por exemplo, podemos usar esse flag para comparar valores, algo como **if(a == 30)**, em linguagem C/C++ ou Java. A comparação afetaria a Zero Flag, mas manteria o valor da variável.
Imagine que o comando "if(a == 30)" pudesse ser reescrito como ***a = a – 30; if(a = 0)...***
Se a variável "a" for realmente igual a 30, depois do cálculo, fazemos alguma coisa, caso contrário teremos perdido nossa variável.

Mas não se preocupe, ao fazer comparações de valores com os registradores, você não irá perder os seus dados.

A – Auxiliar – se este bit for igual a 1 então houve overflow em um resultado desconsiderando o sinal.

P – Paridade – Se P=1, o resultado da operação teve paridade par, caso contrário ímpar

C – Carry – Usado nas operações aritméticas, indica que após uma operação, surgiu um valor superior ao tamanho de um dado.

Por exemplo, se estamos trabalhando com um byte e tentamos fazer uma soma, digamos 250 + 7, nosso byte não vai comportar o resultado – 257 – pois um byte suporta no máximo o valor 255(0xFF) e, para refletir isso, o resultado dessa soma será 2 e a Carry Flag será configurado, isto é, passará a valer 1, indicando que "subiu um" em nossa soma.

Os bits não indicados não são usados e devem conter sempre o valor 0.

Quando fazemos comparações entre valores, os FLAGS são atualizados com o resultado da operação. Através das instruções de desvio condicional, podemos montar o que seria correspondente ao comando “IF” de uma linguagem de alto nível. Por exemplo: o comando em assembly CMP <operando 1>, <operando 2> compara os valores atualizando o registrador FLAGS. Se o operando 1 for igual ao operando 2, então o bit Z possuirá o valor 1, indicando a igualdade.

Mas na maioria das vezes, não precisaremos no preocupar com o valor de cada um desses flags, basta que saibamos como usar as instruções de comparação e de saltos (jumps) condicionais.

As principais instruções de desvio são:

Instruções	Condição de desvio
JÁ	> maior
JAE	>= maior ou igual
JB	< menor
JBE	<= menor ou igual
JE ou JZ	= igual
JG	> maior com sinal
JGE	>= maior igualcom sinal
JL	< menor com sinal
JLE	<= menor ou igual com sinal
JNE ou JNZ	<> diferente

Suponha o seguinte trecho de um programa em C/C++ que não faz absolutamente nada de interessante, mas servirá para analisarmos melhor os bits da FLAG:

```
ax = 0x200;
ax = cx * 0x200;
if (ax == 0x1000)
    ax = 0x3000;
else
ax = 0x2000;
```

Um compilador poderia construir a seguinte versão em Assembly:

```
mov ax, 0x200
mul cx
cmp ax, 0x1000
je _AX_3000
jmp _AX_2000
_AX_3000:
mov ax, 0x3000
jmp _Sai
_AX_2000:
ov ax, 0x2000
_Sai:
```

A instrução **CMP** faz uma comparação entre dois valores e reflete o resultado no registrador de estado do processador.

Outro exemplo em assembly, apresentado abaixo, compara o valor do acumulador AX ao valor 10, se AX for maior que 10, então o controle da execução desvia para o rótulo LOOP1.

CMP AX,10 -> compara AX com o valor 10 e atualiza a FLAG.
JG LOOP1 -> desvia para LOOP1 se AX for maior que 10.

LOOP1: ....

Como pode ser percebido o comando **CMP** atualiza o registrador FLAG e usamos uma das instruções de comparação que implicitamente irá avaliar um dos bits do registrador FLAG.

Embora conhecer qual bit do registrador FLAG é afetado seja importante, não precisamos, obrigatoriamente, saber quais deles foram afetados, pois basta usar uma das instruções definidas para comparação e usá-la para obter um resultado.

As instruções correspondentes aos saltos vão avaliar os bits do registrador FLAG e executar um salto, dependendo do valor encontrado nestes bits.

3) Operações na ULA

Na figura abaixo, demonstra-se com maiores detalhes a CPU e seus componentes internos.

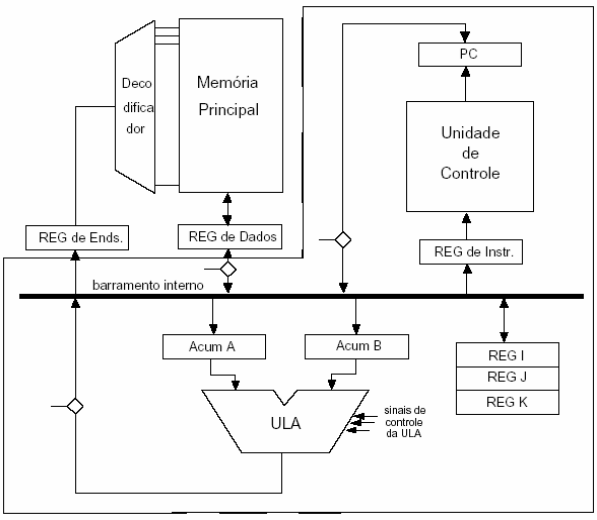

Figura 3: Detalhes da CPU e a memória principal

Para uma operação de soma de dois números, a unidade de controle (UC) se comporta da seguinte maneira em relação ao programa armazenado na memória apresentado abaixo (Reed, 2004).

0: 1000000100000101 // carrega posição de memória 5 em R0
1: 1000000100100110 // carrega posição de memória 6 em R1
2: 1010000100100001 // adiciona R0 e R1, armazena em R2
3: 1000001001000111 // armazena R2 na posição de memória 7
4: 1111111111111111 // parada
5: 0000000000001001 // dado a ser somado: 9
6: 0000000000000001 // dado a ser somado: 1
7: 0000000000000000 // posição onde a soma é armazenada

As primeiras cinco posições de memória (endereços de 0 até 4) contêm instruções em linguagem de máquina para somar dois números e armazenar sua soma de volta na memória. Os números podem ser somados e são armazenados na posição de memória 5 e 6. Para executar este programa, a Unidade de Controle deve seguir os seguintes passos:

1. Primeiro, o contador de programa é inicializado: PC = 0.
2. A instrução na posição de memória 0 (correspondente ao valor atual do PC) é carregada e o PC é incrementado: PC = 0 + 1 = 1.
3. Como esta instrução (1000000100000101) não é uma instrução de parada (HALT), ela é decodificada: o hardware da CPU é configurado de modo que o conteúdo da posição de memória 5 seja carregado no registrador R0 e um ciclo de CPU é executado.
4. A próxima instrução, na posição de memória 1 (correspondente ao valor atual do PC) é carregada e o PC é incrementado: PC = 1 + 1 = 2.
5. Como esta instrução (1000000100100110) não é uma instrução de parada (HALT), ela é decodificada: o hardware da CPU é configurado de modo que o conteúdo da posição de memória 6 seja carregado no registrador R1 e um ciclo de CPU é executado.
6. A próxima instrução, na posição de memória 2 (correspondente ao valor atual do PC) é carregada e o PC é incrementado: PC = 2 + 1 = 3.
7. Como esta instrução (1010000100100001) não é uma instrução de parada (HALT), ela é decodificada: o hardware da CPU é configurado de

modo que o conteúdo do registrador R0 é somado ao conteúdo do registrador R1 e o resultado é armazenado no registrador R2 e um ciclo de CPU é executado.

8. A próxima instrução, na posição de memória 3 (correspondente ao valor atual do PC) é carregada e o PC é incrementado: PC = 3 + 1 = 4.
9. Como esta instrução (1000001001000111) ainda não é uma instrução HALT, ela é decodificada: o hardware da CPU é configurado de modo que o conteúdo do registrador R2 seja armazenado na posição de memória 7 e um ciclo de CPU é executado.
10. A próxima instrução, na posição de memória 4 (correspondente ao valor atual do PC) é carregada e o PC é incrementado: PC = 4 + 1 = 5.

Como esta instrução (1111111111111111) é uma instrução HALT, a Unidade de Controle reconhece o fim do programa e pára a execução.

Como pôde perceber, os programas escritos em linguagem de montagem têm que ser mais bem detalhado e para isto é necessário conhecer os componentes, o fluxo de informação e as instruções que ativam estes componentes. No exemplo dado acima sobre a soma de dois números, vários passos tiveram que ser feitos para obter o resultado, você deve ter percebido que o programador deve saber onde os dados estão armazenados na memória e onde os resultados serão armazenados. No exemplo dado os comandos estão apresentados em binário, mas para dar maior visibilidade ao programador, as linguagens de montagem usam o formato hexadecimal e abreviações de comandos.

Referências

Reed D. **A Balanced Introduction to Computer Science and Programming** tradução do capítulo 14. Creighton University**,** Prentice Hall, 2004. disponível em http://professores.faccat.br/assis/davereed/14-DentroDoComputador.html

Responsável pelo Conteúdo:
Prof. Ms. Vagner da Silva
Revisão Textual:
Profª Ms. Rosemary Toffoli

www.cruzeirodosul.edu.br

Campus Liberdade

Rua Galvão Bueno, 868

01506-000

São Paulo SP Brasil

Tel: (55 11) 3385-3000