# Sistemas Paralelos — Lab — Parte 1

Carlos Antônio Ruggiero

agosto de 2020

## 1 Introdução

Segundo Amdahl, **arquitetura** pode ser definida como o conjunto de características do computador visto pelo programador em linguagem de máquina. Dois computadores com mesma arquitetura, portanto, devem ser capazes de executar os mesmos programas.

O conceito de arquitetura surge para possibilitar aos projetistas a incorporação de novos circuitos e equipamentos mais rápidos aos computadores, tão logo estes estejam disponíveis, sem que haja a necessidade de se reescrever programas básicos como sistemas operacionais, compiladores, aplicativos importantes, etc.

A primeira arquitetura, com esse nome, da história da computação foi o sistema 360/370 da IBM, que durou várias décadas, englobando diversas fases de tecnologia. Outra arquitetura muito bem sucedida foi a do PDP-11 da Digital e arquitetura VAX-11, também da Digital.

Na atualidade, a arquitetura mais usada no mundo é a chamada **x86** da empresa americana Intel. Também conhecida como **ia86**, ela é utilizada desde o início da década de oitenta em computadores do tipo **IBM PC**. Outra arquitetura que ganhou muita importância nos últimos anos é a ARM, muito utilizada em sistemas embarcados, *smartphones* e *tablets*.

## 2 A Arquitetura x86 - Histórico

Desde o início da computação há mais de cinco décadas, tem-se notado que o avanço da tecnologia de hardware é muito mais acelerado que o correspondente avanço no software. Assim, o conceito de arquitetura é extremamente útil para os fabricantes de computadores que definem uma arquitetura e a mantém por muitos anos (ou décadas).

Na década de 70, surgiram os microprocessadores monolíticos, ou seja, UCPs completas em uma única pastilha. Este fato teve um enorme impacto na tecnologia de computadores e permitiu o surgimento do computador pessoal (**PC**). Infelizmente, porém, os fabricantes de microprocessadores não se preocuparam em definir arquiteturas estáveis, que pudessem durar décadas, como era o caso com computadores de grande porte. Arquiteturas diferentes se sucediam com grande velocidade, tentando aproveitar os rápidos avanços na tecnologia de produção de circuitos integrados.

A Intel, pioneira na área de microprocessadores, projetou várias arquiteturas, tais como a 4004, 8004, 8080, 8085, etc. Seu primeiro microprocessador de 16 bits foi o 8086, lançado no final da década de 70.

No início da década de 80, a IBM resolveu produzir um microcomputador barato, que pudesse ser adquirido por pessoas comuns e utilizados em tarefas pessoais e rotineiras. A esperança era criar um grande mercado de computadores de uso pessoal que ultrapasse a casa de 1 milhão de unidades vendidas. O microprocessador escolhido foi o 8086 da Intel (na verdade, foi o 8088, com mesma arquitetura do 8086 mas com barramento externo de apenas 8 bits). Surgia então o **IBM PC**.

O enorme sucesso do IBM PC acabou forçando a Intel a perpetuar a arquitetura do 8086 para não prejudicar o grande mercado de software surgido para tal computador. Assim, até os dias atuais, programadores são obrigados a utilizar uma arquitetura que não foi projetada para durar tanto tempo. A arquitetura x86, quando comparada com arquiteturas mais modernas, apresenta grandes deficiências, sendo de difícil compreensão. O gigantesco número de computadores com essa arquitetura porém, justifica o esforço no seu aprendizado.

A arquitetura x86 apresentou evoluções durante os anos, mas sempre manteve a compatibilidade com o pioneiro 8086. As principais mudanças ocorreram com a introdução do 386, que foi o primeiro microprocessador de 32 bits da família.

# 3 Visão Geral

O conjunto de registradores de **uso geral** do 8086 pode ser visto na figura 1.

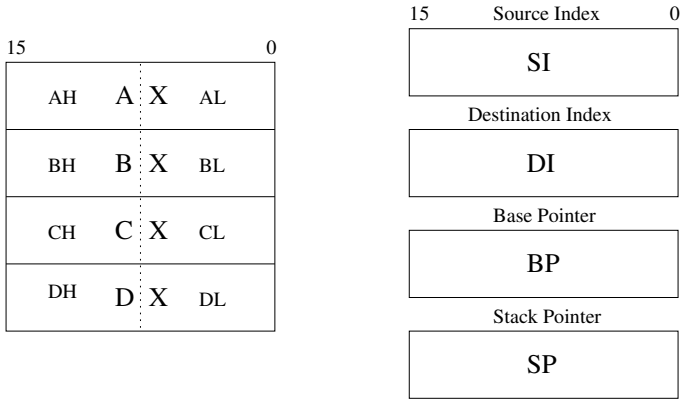


Figura 1: Registradores do 8086.

Todos os registradores na figura são de 16 bits. Além desses registradores de uso geral, tem-se 4 registradores de endereçamento, conhecidos como **registradores de segmento**. A figura 2 mostra tais registradores.

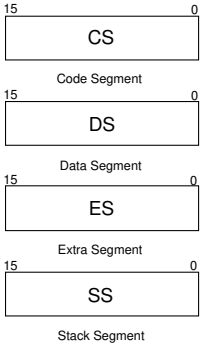


Figura 2: Registradores de segmento do 8086.

Assim como os registradores de uso geral, os registradores de segmento são de 16 bits. Eles devem ser usados, exclusivamente, para manipulação de endereços.

Finalmente, é necessário citar o registrador de **flags**, que mantém o estado atual do processador, e o contador de programa chamado de **IP**.

Com a introdução do 386, os registradores passam a ter 32 bits, e, ao nome original adiciona-se um "e" (de *extended*). Assim, por exemplo, o antigo registrador BX passa a se chamar, EBX, o SP passa para ESP e assim por diante. Veja a figura 3.

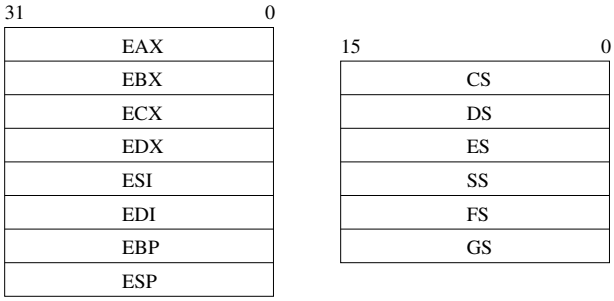


Figura 3: Registradores do 386

Com a chegada do x86-64, os registradores passam a ser de 64 bits e, quando se quer utilizar o comprimento todo, adiciona-se um *r* ao invés do *e*. Assim, têm-se *rax*, *rbx*, etc.. Agora, porém, são adicionados mais 8 registradores de 64 bits de propósito geral, chamados de *r8* a *r15*. Os 32, 16 e 8 bits menos significativos desses novos registradores são referenciados por *r8d*, *r8w* e *r8b* (por exemplo).

Os registradores anteriores (AH, AL, AX, BH, etc) ainda podem ser acessados, de forma que programas escritos para o 8086 ainda possam ser executados nos processadores mais novos. Os registradores de 64 bits podem ser vistos na figura 4.

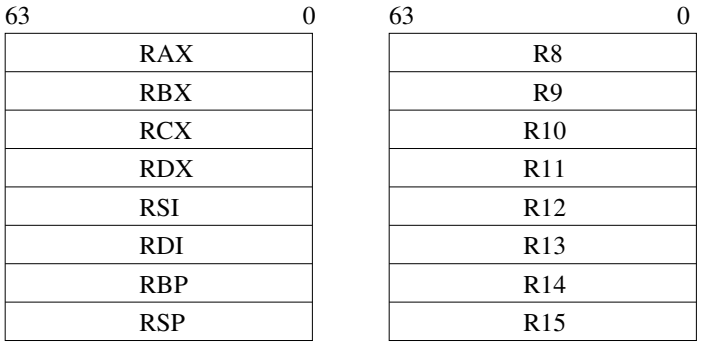


Figura 4: Registradores do x86-64

# 4 Tipos de Dados

Os tipos de dados básicos da arquitetura x86 são: **byte**, **word**, **doubleword** e **quadword**, com tamanhos de 8, 16, 32 e 64 bits, respectivamente.

Esses dados não precisam estar alinhados na memória, mas, se estiverem, propiciam acesso mais rápido por parte do processador.

Algumas instruções são capazes de manipular outros tipos de dados além dos básicos já mencionados. O tipo **inteiro com sinal** é muito usado em operações aritméticas e é sempre interpretado em

complemento de dois. Todos os tipos básicos, com exceção do quadword, podem ser interpretados como inteiros com sinal, sendo o sinal definido pelos bits 7, 15 e 31.

Também estão disponíveis os inteiros sem sinal nos formatos de bytes, words e doublewords. Este tipo de dado é também chamado pela Intel de **ordinal**.

Inteiros codificados como **BCD** são inteiros de 4 bits sem sinal que podem assumir valores entre 0 e 9. Duas variantes estão disponíveis: a **unpacked**, onde cada dígito ocupa um byte, e a **packed**, onde um byte abriga dois dígitos.

# 5 Parte Prática

Execute as seguintes atividades em seu computador. As atividades a serem executadas estão enumeradas.

1. Entre em seu computador (se você não tiver conta, peça auxílio).
2. Abra um terminal do Linux.
3. Certifique-se de ter bom conhecimento de um dos editores do linux: o vi (VIM) ou o emacs.
4. Com seu editor favorito crie um arquivo chamado **hello.asm** que tenha o seguinte conteúdo:

```
section .data                           ;seção de dados

msg     db      "Oi gente!",0xa         ;cadeia de carateres  na memória
len     equ     $ - msg                 ;comprimento da cadeia

section .text                           ;seção de código

   global _start                        ;Entrada para o linkeditor

_start:

; Chamada do sistema (Linux) para escrever um caracter.

       mov     rax,1                    ;Qual chamada? sys_write.
       mov     rdi,1                    ;Onde escrever? - Primeiro argumento
       mov     rdx,len                 ;Taman. da mensagem? - Terc argumento
       syscall                    ;Vai!

;Chamada do sistema - acabar (sys_exit)

       mov     rax,60                    ;Qual chamada? sys_exit.
       mov     rdi,0                    ;Código de retorno - único argumento
       syscall                     ;Vai!
```

5. Compile o programa com **nasm -f elf64 hello.asm .** Gere o programa executável com **ld -s -o hello hello.o** e execute o programa com **./hello**. O que ocorre?
6. Observe o seguinte no programa:

- Cada instrução pode apresentar 4 campos: o primeiro (opcional) mais a direita é o "rótulo" que indica a endereço de memória da instrução; o segundo é a instrução propriamente dita seguida pelo campo de argumentos. FInalmente temos o campo opcional dos comentários.
- Na liguagem montadora aceita pelo **nasm,** podemos definir partes do programa que conterão código (com a *diretiva section*, como em **section .text**) e partes que conterão dados(**section .data**). É possível ainda definir área para a pilha (**section .stack**).
- São definidas pseudo-instruções as quais **não** são instruções da arquitetura mas servem para guiar o montador na geração de código. Por exemplo, a pseudo-instrução **db** reserva um ou mais bytes na memória e atribui a esses bytes os valores especificados. No programa apresentado, são atribuídos os valores ASCII dos caracteres da cadeia “Oi gente!" às posições de memória iniciadas em “msg".
- A pseudo-instrução **equ** define um valor para uma constante. No exemplo, **len** recebe o valor do comprimento da cadeia. Repare que $ representa **a posição atual da memória**. Assim, $ - MSG será o comprimento, em bytes, da cadeia "OI MUNDO!".
- A pseudo-instrução **global** indica que o símbolo definido pode ser acessado por módulos externos. No exemplo, o símbolo **_start** marca o início do código.
- Talvez a instrução mais popular da arquitetura x86 seja a **mov**. Ela copia os dados de uma posição de memória ou registrador para outra posição ou registrador. Lembre-se que a cópia vai do **operando da direita para o da esquerda no NASM.**
- Chamadas do sistema Linux (e FreeBSD, NetBSD, etc), em 64 bits, são efetuadas através da instrução **syscall**. Memorize que o conteúdo do registrador RAX leva o código da operação a ser efetuada pelo sistema. Assim, 01 é uma operação de escrita e 60 é a finalização do programa com retorno ao sistema operacional.

7. Digite o seguinte programa (**hello.S**), monte com **as -o hello.o hello.S**, gere o executável com **ld -s -o hello hello.S** e execute com **./hello**

```
.data                                   #seção de dados

msg:    .ascii      "Oi gente!\n"       #cadeia de carateres definida na memória
        len= . - msg                    #comprimento da cadeia

.text                                   #seção de código

    .global _start      # Ponto de entrada para o linkeditor

_start:

# Chamada do sistema (Linux) para escrever um caracter.

       movq    $1,%rax                  #Qual chamada? sys_write.
       movq    $1,%rdi                  #Onde escrever? Primeiro argumento
       movq    $msg,%rsx                #Qual mensagem? - segundo argumento
       movq    $len,%rdx               #Taman da mensagem? - terceiro argumento
       syscall                    #Vai!
```

```
#Chamada do sistema - acabar (sys_exit)

       movq    $60,%rax                 #Qual chamada? sys_exit.
       movq    $0,%rdi                 #Código de retorno - único argumento
       syscall                         #Vai!
```

8. Observe o seguinte:

   - Este programa foi escrito com a sintaxe do montador **gas** da GNU, que é diferente da sintaxe do nasm.
   - As seções de dado e de código não usam a palavra **section**.
   - Algumas instruções, como o **mov** tem que especificar o tamanho do dado ou seja **movq** (note o q de *quad* no final)
   - Valores imediatos devem ser precedidos pelo **$** e todos os registradores são precedidos por **%** como em **%rax**.
   - Ao contrário do NASM, a direção é da esquerda para a direita: assim, **movq $1,%rax**, coloca o valor imediato 1 no registrador **rax**.
   - O diretivo **EQU** do NASM é substituido pelo sinal de igual como em **len= . - msg**. Note também que a posição atual de montagem é representada por **.** e não por **$**.

# 6 Leitura recomendada

Como tarefa de casa, leia o *Forward* do livro *The Art of Assembly Language Programming*. Se tiver dúvidas de arquiteturas I, leia também os 3 primeiros capítulos.