Assembly x86

ICMC USP SÃO CARLOS

O que, e por que?

O que?

- Assembly é o nome dado para as linguagens de programação que rodam diretamente no processador.
  - Cada arquitetura de processador tem seu próprio assembly.

- A linguagem é composta de instruções simples (ADD, SUB, …)

- x86 eh o mais usado pelos processadores Intel e AMD.

Por que?

- Cada instrução equivale a um caminho pelo qual os dados podem passar, assim que o processador consegue ~processar~
- Todos os codigos Assembly podem ser traduzidos diretamente para binarios, e traduzidos diretamente de volta.
- Se voce é capaz de entender o codigo Assembly, voce consegue entender qualquer programa.

Problemas

- Confuso.

Problemas

- Confuso.
  - Muito.

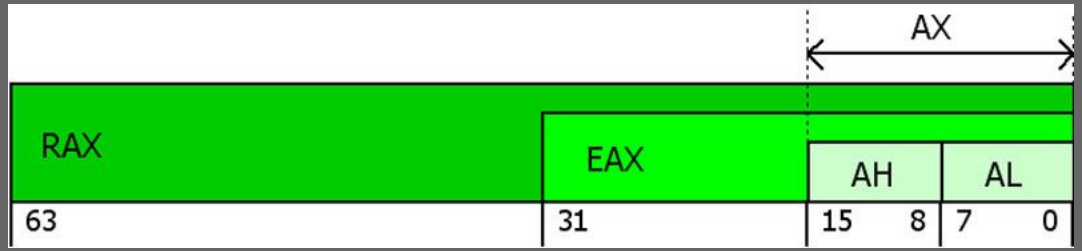
Problemas

- Confuso.
  - Muito.

- Restrições:
  - Operações devem ser realizadas entre registradores.
  - deve-se especificar endereços de memória
  - É necessario programar para cada arquitetura separadamente.

Registradores

- Funcionam equivalentemente as variaveis em C, mas os nomes nao podem ser escolhidos
  - Proposito geral:
    - A, B, C, D
  - gerencia de memoria:
    - ebp, esp

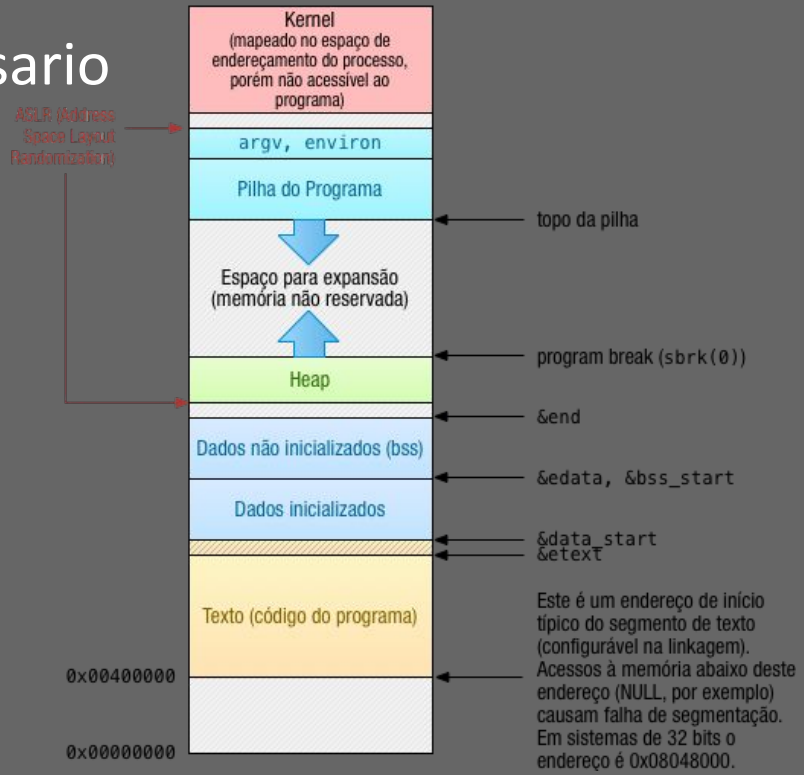


Fonte: https://www.clubedohardware.com.br/artigos/processadores/arquitetura-de-64-bits-da-amd-x86-64-r33920/?nbcpage=3

Organização de programas

Para entender um programa é necessario entender o endereçamento na memória.

fonte:
http://www.ppgia.pucpr.br/~laplima/ensino/s o/materia/03_memoria.html



Alinhamento de instruções

Para acelerar o processamento, endereços de memória são alinhados conforme o tamanho do dado

- inteiros: 4 bytes => alinhado em endereços multiplos de 4
- char: 1 byte => alinhado em qualquer endereço (multiplo de 1)
- Instruções: 8 bytes (64bits) também precisam ser alinhadas:
  - NOP, NOPL, NOPW

Exemplos de codigo

Após compilar o programa, podemos ver o assembly usando:
objdump -d <prog>

```
int main(){
    printf("Hello World\n");
    return 0;
}
```

```
push %rbp
mov %rsp,%rbp
lea 0x9f(%rip), %rdi
callq 510 <puts@plt>
mov $0x0, %eax
pop %rbp
retq
nopw %cs:0x0(%rax,%rax,1)
nopl 0x0(%rax,%rax,1)
```

Exemplos de codigo

```
int main(){
    int a,b,c;
    a=5;
    b=2;
    c = a+b;
    return 0;
}
```

```
push %rbp
mov %rsp, %rbp
movl $0x5, -0xc(%rbp)
movl $0x2, -0x8(%rbp)
mov  -0xc(%rbp), %edx
mov -0x8(%rbp), %eax
add %edx, %eax
mov %eax, -0x4(%rbp)
mov 0x0, %eax
pop %rbp
retq
xchg %ax, %ax
```

Exemplos de codigo

```
int main(int argc, char** argv){
        char s[20];
        cpy(argv[1],s,20);
}
```

```
push %rbp
mov %rsp, %rbp
sub 0x30, %rsp
mov %edi, -0x24(%rbp)
mov %rsi, -0x30(%rbp)
mov %fs: 0x28,%rax

mov %rax, -0x8(%rbp)
xor %eax,%eax
mov -0x30(%rbp), %rax
add $0x8, %rax
mov (%rax),%rax
lea -0x20(%rbp), %rdx
mov $0x0,%eax
mov %rcx,%rsi
mox %rax,%rdi
callq <cpy>
```

```
mov $0x0, %eax
mov -0x8(%rbp), %rdx
xor %fs: 0x28, %rdx

je 706 <main+0x56>
callq 540 <__stack_chk_fail@plt>
leaveq
retq
nopl 0x0(%rax,%rax,1)
```

Exemplos de codigo

```
void cpy(char *src, char*
dest,  int size){

        int i;
        for(i=0;i<size;i++)
                dest[i] = src[i];

}
```

```
        push %rpb
        mov %rsp, %rbp
        mov %rdi, -0x18(%rbp)
        mov %rsi, -0x20(%rbp)
        mov %edx, -0x24(%rbp)
        movl $0x0, -0x4(%rbp)
682     jmp 6a5 <cpy+0x3b>
        mov -0x4(%rbp), %eax
        movslq %eax, %rdx
        mov -0x18(%rbp), %rax
        add %rdx, %rax
        mov -0x4(%rbp), %rdx
        movslq %edx, %rcx
        mov -0x20(%rbp), %rdx
        add %rcx, %rdx
        movzbl (%rax),%eax
        mov %al, (%rdx)
        addl $0x1, -0x4(%rbp)
6a5     mov -0x4(%rbp), %eax
        cmp -0x24(%rbp), %eax
        jl 682 <cpy+0x18>
        nop
        pop %rbp
        retq
```