

Estrutura do tema ISA do IA-32

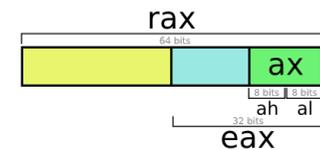
1. Desenvolvimento de programas no IA-32 em Linux
2. Acesso a operandos e operações
3. Suporte a estruturas de controlo
4. Suporte à invocação/regresso de funções
5. Análise comparativa: IA-32, x86-64 e MIPS (RISC)
6. Acesso e manipulação de dados estruturados

x86-64 Integer Registers

%rax	%eax	%r8	%r8d
%rbx	%ebx	%r9	%r9d
%rcx	%ecx	%r10	%r10d
%rdx	%edx	%r11	%r11d
%rsi	%esi	%r12	%r12d
%rdi	%edi	%r13	%r13d
%rsp	%esp	%r14	%r14d
%rbp	%ebp	%r15	%r15d

- Twice the number of registers
- Accessible as 8, 16, 32, 64 bits

University of Washington



x86-64 Integer Registers: Usage Conventions

%rax	Return value	%r8	Argument #5
%rbx	Callee saved	%r9	Argument #6
%rcx	Argument #4	%r10	Caller saved
%rdx	Argument #3	%r11	Caller Saved
%rsi	Argument #2	%r12	Callee saved
%rdi	Argument #1	%r13	Callee saved
%rsp	Stack pointer	%r14	Callee saved
%rbp	Callee saved	%r15	Callee saved

Funções em assembly: IA-32 versus Intel 64 (1)

Principal diferença na implementação de funções:

- na organização dos registos
 - IA-32: poucos registos genéricos (só 6) => variáveis locais em reg e argumentos na *stack*
 - Intel 64: 16 registos genéricos => mais registos para variáveis locais & para passagem e uso de argumentos (8 + 6)
- consequências:
 - menor utilização da *stack* na arquitetura Intel 64
 - Intel 64 potencialmente mais eficiente

Análise de um exemplo (swap) ...

Revisão da codificação de swap e call_swap no IA-32

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

```
void call_swap()
{
    int zip1 = 15213;
    int zip2 = 91125;
    swap(&zip1, &zip2);
}
```

```
_swap:
    pushl %ebp
    movl %esp, %ebp
    pushl %ebx

    movl 12(%ebp), %ecx
    movl 8(%ebp), %edx
    movl (%ecx), %eax
    movl (%edx), %ebx
    movl %eax, (%edx)
    movl %ebx, (%ecx)

    movl -4(%ebp), %ebx
    movl %ebp, %esp
    popl %ebp
    ret
```

```
_call_swap:
    pushl %ebp
    movl %esp, %ebp
    subl $24, %esp

    movl $15213, -4(%ebp)
    movl $91125, -8(%ebp)
    leal -4(%ebp), %eax
    movl %eax, (%esp)
    leal -8(%ebp), %eax
    movl %eax, 4(%esp)
    call _swap

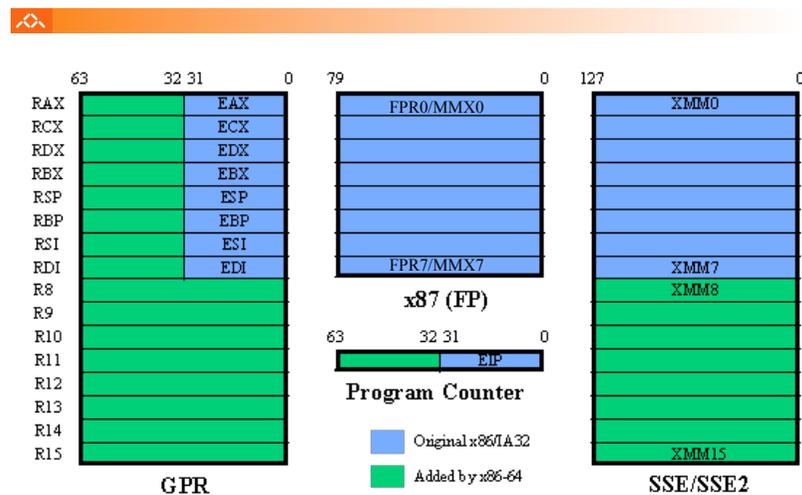
    movl %ebp, %esp
    popl %ebp
    ret
```

Funções em assembly: IA-32 versus Intel 64 (2)

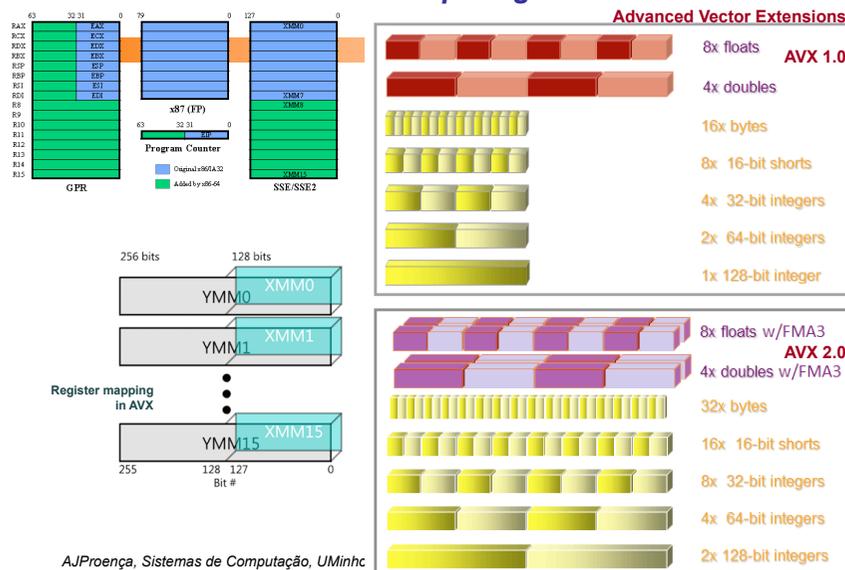
IA-32	Intel 64
<pre> _swap: pushl %ebp movl %esp, %ebp pushl %ebx movl 8(%ebp), %edx movl 12(%ebp), %ecx movl (%edx), %ebx movl (%ecx), %eax movl %eax, (%edx) movl %ebx, (%ecx) popl %ebx popl %ebp ret _call_swap: pushl %ebp movl %esp, %ebp subl \$24, %esp movl \$15213, -4(%ebp) movl \$91125, -8(%ebp) leal -4(%ebp), %eax movl %eax, (%esp) leal -8(%ebp), %eax movl %eax, 4(%esp) call _swap movl %ebp, %esp popl %ebp ret </pre>	<pre> swap: pushq %rbp movq %rsp, %rbp movl (%rdi), %eax movl (%rsi), %ecx movl %ecx, (%rdi) movl %eax, (%rsi) popq %rbp retq call_swap: pushq %rbp movq %rsp, %rbp subq \$16, %rsp movl \$15213, -4(%rbp) movl \$91125, -8(%rbp) leaq -4(%rbp), %rdi leaq -8(%rbp), %rsi callq _swap addq \$16, %rsp popq %rbp retq </pre>
Total: 63 bytes	Total: 54 bytes

Total de acessos à stack: 14 no IA-32, 6 no Intel 64 !

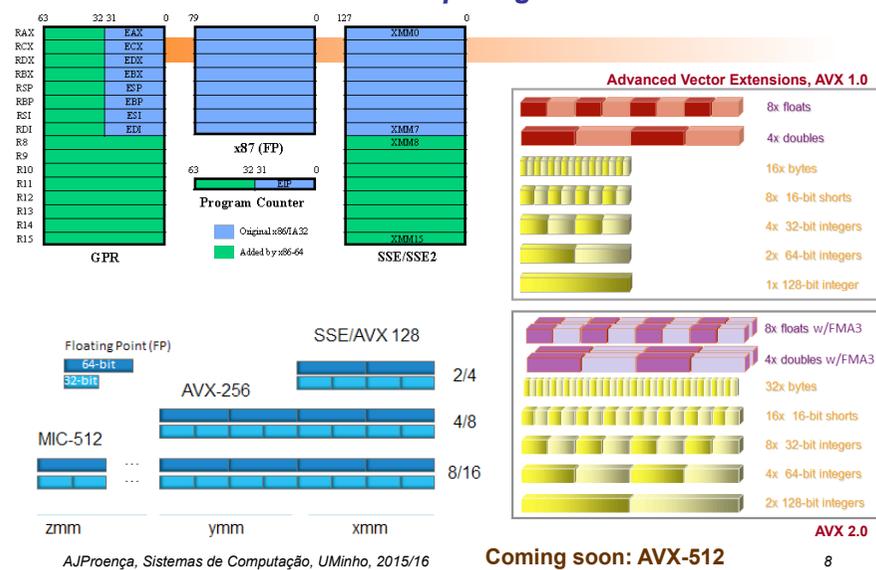
Vector computing extensions in Intel 64



Vector computing extensions in Intel 64



Vector computing extensions in Intel 64



Coming soon: AVX-512

Caracterização das arquiteturas RISC

- conjunto reduzido e simples de instruções
- operandos sempre em registos
- formatos simples de instruções
- modos simples de endereçamento à memória
- uma operação elementar por ciclo máquina

Ex de uma arquitetura RISC:

ARM



RISC versus IA-32 :

- RISC: conjunto reduzido e simples de instruções
 - pouco mais que o subset do IA-32 já apresentado...
 - instruções simples, mas eficientes
- operações aritméticas e lógicas:
 - 3-operandos (RISC) versus 2-operandos (IA-32)
 - RISC: operandos sempre em registos, 32 registos genéricos visíveis ao programador, sendo normalmente
 - 1 reg apenas de leitura, com o valor 0
 - 1 reg usado para guardar o endereço de regresso da função
 - 1 reg usado como stack pointer (convenção do s/w)

RISC versus IA-32 (cont.):

- RISC: modos simples de endereçamento à memória
 - apenas 1 modo de especificar o endereço:
 $Mem[C^{te} + (Reg_b)]$ OU $Mem[(Reg_b) + (Reg_i)]$
 - 2 ou 3 modos de especificar o endereço:
 $Mem[C^{te} + (Reg_b)]$ e/ou
 $Mem[(Reg_b) + (Reg_i)]$ e/ou
 $Mem[C^{te} + (Reg_b) + (Reg_i)]$
- RISC: uma operação elementar por ciclo máquina
 - por ex. push/pop (IA-32) substituído pelo par de instruções sub&store/load&add (RISC)

RISC versus IA-32 (cont.):

- RISC: formatos simples de instruções
 - comprimento fixo e poucas variações
 - ex.: MIPS

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

Funções em assembly:
IA-32 versus MIPS (RISC) (1)

Revisão da codificação
de swap e call_swap no IA-32

Principal diferença na implementação de funções:

- na organização dos registos
 - IA-32: poucos registos genéricos => variáveis e argumentos normalmente na *stack*
 - RISC: 32 registos genéricos => mais registos para variáveis locais, & registos para passagem de argumentos & registo para endereço de regresso
- consequências:
 - menor utilização da *stack* nas arquiteturas RISC
 - RISC potencialmente mais eficiente

Análise de um exemplo (swap) ...

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

```
void call_swap()
{
    int zip1 = 15213;
    int zip2 = 91125;
    swap(&zip1, &zip2);
}
```

```
_swap:
    pushl %ebp
    movl %esp, %ebp
    pushl %ebx

    movl 12(%ebp), %ecx
    movl 8(%ebp), %edx
    movl (%ecx), %eax
    movl (%edx), %ebx
    movl %eax, (%edx)
    movl %ebx, (%ecx)

    movl -4(%ebp), %ebx
    movl %ebp, %esp
    popl %ebp
    ret
```

```
_call_swap:
    pushl %ebp
    movl %esp, %ebp
    subl $24, %esp

    movl $15213, -4(%ebp)
    movl $91125, -8(%ebp)
    leal -4(%ebp), %eax
    movl %eax, (%esp)
    leal -8(%ebp), %eax
    movl %eax, 4(%esp)
    call _swap

    movl %ebp, %esp
    popl %ebp
    ret
```

Funções em assembly:
IA-32 versus MIPS (RISC) (2)

Funções em assembly:
IA-32 versus MIPS (RISC) (3)

IA-32	MIPS
<pre>_swap: pushl %ebp movl %esp, %ebp pushl %ebx movl 8(%ebp), %edx movl 12(%ebp), %ecx movl (%edx), %ebx movl (%ecx), %eax movl %eax, (%edx) movl %ebx, (%ecx) popl %ebx popl %ebp ret _call_swap: pushl %ebp movl %esp, %ebp subl \$24, %esp movl \$15213, -4(%ebp) movl \$91125, -8(%ebp) leal -4(%ebp), %eax movl %eax, (%esp) leal -8(%ebp), %eax movl %eax, 4(%esp) call _swap movl %ebp, %esp popl %ebp ret</pre> <p>Total: 63 bytes</p>	<pre>swap: lw \$v1, 0(\$a0) lw \$v0, 0(\$a1) sw \$v0, 0(\$a0) sw \$v1, 0(\$a1) j \$ra call_swap: subu \$sp, \$sp, 32 sw \$ra, 24(\$sp) li \$v0, 15213 sw \$v0, 16(\$sp) li \$v0, 0x10000 ori \$v0, \$v0, 0x63f5 sw \$v0, 20(\$sp) addu \$a0, \$sp, 16 # &zip1= sp+16 addu \$a1, \$sp, 20 # &zip2= sp+20 jal swap lw \$ra, 24(\$sp) addu \$sp, \$sp, 32 j \$ra</pre> <p>Total: 72 bytes</p>

call_swap

1. Invocar swap

- salvaguardar registos
- passagem de argumentos
- chamar rotina e guardar endereço de regresso

IA-32

Não há reg para salvag.
Calcula &zip2
Push &zip2
Calcula &zip1
Push &zip1
Invoca swap

Acessos à stack

MIPS

sw \$ra, 24(\$sp) Salvag. reg c/ endereço regresso
addu \$a0, \$sp, 16 Calcula & coloca &zip1 no reg arg 0
addu \$a1, \$sp, 20 Calcula & coloca &zip2 no reg arg 1
jal swap Invoca swap

Funções em assembly:
IA-32 versus MIPS (RISC) (4)

swap

1. Inicializar swap

- atualizar frame pointer
- salvar registos
- reservar espaço p/ locais

IA-32

```
swap:
  pushl %ebp      Salvag. antigo %ebp
  movl  %esp, %ebp %ebp novo frame pointer
  pushl %ebx      Salvag. %ebx
```

Acessos à stack

Não é preciso espaço p/ locais

MIPS

Frame pointer p/ atualizar: **NÃO**
Registos p/ salvar: **NÃO**
Espaço p/ locais: **NÃO**

Funções em assembly:
IA-32 versus MIPS (RISC) (5)

swap

2. Corpo de swap ...

IA-32

```
movl 12(%ebp), %ecx Coloca yp em reg
movl 8(%ebp), %edx  Coloca xp em reg
movl (%ecx), %eax   Coloca y em reg
movl (%edx), %ebx   Coloca x em reg
movl %eax, (%edx)   Armazena y em *xp
movl %ebx, (%ecx)   Armazena x em *yp
```

Acessos à memória (todas...)

MIPS

```
lw $v1, 0($a0) Coloca x em reg
lw $v0, 0($a1) Coloca y em reg
sw $v0, 0($a0) Armazena y em *xp
sw $v1, 0($a1) Armazena x em *yp
```

Funções em assembly:
IA-32 versus MIPS (RISC) (6)

swap

3. Término de swap ...

- libertar espaço de var locais
- recuperar registos
- recuperar antigo frame pointer
- regressar a call_swap

IA-32

```
popl %ebx      Não há espaço a libertar
              Recupera %ebx
movl %ebp, %esp Recupera %esp
popl %ebp      Recupera %ebp
ret            Regressa à função chamadora
```

Acessos à stack

MIPS

Espaço a libertar de var locais: **NÃO**
Recuperação de registos: **NÃO**
Recuperação do frame ptr: **NÃO**
Regressa à função chamadora

```
j $ra
```

Funções em assembly:
IA-32 versus MIPS (RISC) (7)

call_swap

2. Terminar invocação de swap ...

- libertar espaço de argumentos na stack...
- recuperar registos

IA-32

```
addl $8, (%esp) Atualiza stack pointer
              Não há reg's a recuperar
```

Acessos à stack

MIPS

```
lw $ra, 24($sp) Espaço a libertar na stack: NÃO
              Recupera reg c/ ender regresso
```

Total de acessos à stack: 14 no IA-32, 6 no MIPS !