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 *vs.* x86-64 e RISC (MIPS e ARM)
6. Acesso e manipulação de dados estruturados

Principal diferença na organização interna:

– 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

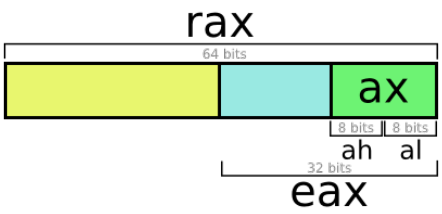
Análise de um exemplo (`swap`) ...

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

Principal diferença na organização interna:

- 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;
}
```

```
_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
```

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

```
_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

```
_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
```

Total: 63 *bytes*

Intel 64

```
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
```

Total: 54 *bytes*

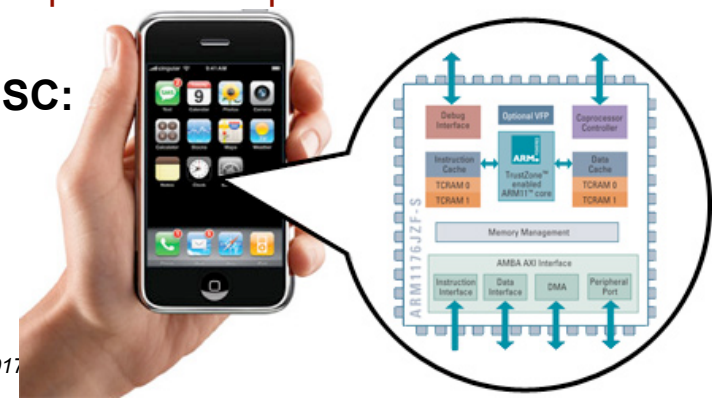
Total de acessos à *stack*: 15 no IA-32, 9 no Intel 64 !

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 muito eficientes em *pipeline*
- operações aritméticas e lógicas:
  - 3-operandos (RISC) *versus* 2-operandos (IA-32)
  - RISC: operandos sempre em registos,
    16/32 registos genéricos visíveis ao programador, sendo normalmente
    - 1 reg apenas de leitura, com o valor 0 (em 32 registos)
    - 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[Cte+(Regb)]` ou `Mem[(Regb)+(Regi)]`
  - ou poucos modos de especificar o endereço:
    `Mem[Cte+(Regb)]` e/ou
    `Mem[(Regb)+(Regi)]` e/ou
    `Mem[Cte+(Regb)+(Regi)]`
- 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

  - ex.: ARM

Name	Format	Example								Comments
Field size		4 bits	2 bits	1 bit	4 bits	1 bit	4 bits	4 bits	12 bits	All ARM instructions are 32 bits long
DP format	DP	Cond	F	I	Opcode	S	Rn	Rd	Operand2	Arithmetic instruction format
DT format	DT	Cond	F	Opcode			Rn	Rd	Offset12	Data transfer format
Field size		4 bits	2 bits	2 bits	24 bits					
BR format	BR	Cond	F	Opcode	signed_immed_24					B and BL instructions

- ARM: the most popular embedded core
- Similar basic set of instructions to MIPS

	ARM	MIPS
Date announced	1985	1985
Instruction size	32 bits	32 bits
Address space	32-bit flat	32-bit flat
Data alignment	Aligned	Aligned
Data addressing modes	9	3
Registers	15 × 32-bit	31 × 32-bit
Input/output	Memory mapped	Memory mapped

Chapter 2 — Instructions: Language of the Computer — 78

11

IA-32 versus RISC (1)

Principal diferença na organização interna:

– organização dos registos

- IA-32: poucos registos genéricos =>
  variáveis e argumentos normalmente na *stack*
- RISC: 16/32 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`) ...

Convenção na utilização de registos RISC: MIPS e ARM

MIPS

Name	Register	Usage
$zero	$0	Always 0 (forced by hardware)
$at	$1	Reserved for assembler use
$v0 – $v1	$2 – $3	Result values of a function
$a0 – $a3	$4 – $7	Arguments of a function
$t0 – $t7	$8 – $15	Temporary Values
$s0 – $s7	$16 – $23	Saved registers (preserved across call)
$t8 – $t9	$24 – $25	More temporaries
$k0 – $k1	$26 – $27	Reserved for OS kernel
$gp	$28	Global pointer (points to global data)
$sp	$29	Stack pointer (points to top of stack)
$fp	$30	Frame pointer (points to stack frame)

ARM

Name	Register number	Usage	Preserved on call?
a1-a2	0–1	Argument / return result / scratch register	no
a3-a4	2–3	Argument / scratch register	no
v1-v8	4–11	Variables for local routine	yes
ip	12	Intra-procedure-call scratch register	no
sp	13	Stack pointer	yes
lr	14	Link Register (Return address)	yes
pc	15	Program Counter	n.a.

IA-32 versus RISC (2)

Principal diferença na organização interna:

- organização dos registos
  - IA-32: poucos registos genéricos => variáveis e argumentos normalmente na *stack*
  - RISC: 16/32 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`) ...

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;
}
```

```
_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
```

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

```
_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)

IA-32

```
_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
```

Total: 63 *bytes*

MIPS

```
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
```

Total: 72 *bytes*

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

call_swap

1. Invocar `swap`
- salvaguardar registos
- passagem de argumentos
- chamar rotina e guardar endereço de regresso

IA-32

```
                          Não há reg para salvag.
leal   -4(%ebp),%eax      Calcula &zip2
pushl  %eax               Push &zip2
leal   -8(%ebp),%eax      Calcula &zip1
pushl  %eax               Push &zip1
call   swap               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*
- salvaguardar registos
- reservar espaço p/ locais

IA-32

```
swap:
  pushl %ebp        Salvag. antigo %ebp
  movl  %esp,%ebp   %ebp novo frame pointer
  pushl %ebx        Salvag. %ebx
                    Não é preciso espaço p/ locais
```

Acessos à *stack*

MIPS

Frame pointer p/ actualizar:	NÃO
Registos p/ salvaguardar:	NÃO
Espaço p/ locais:	NÃO

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
```


swap

3. Término de `swap` ...

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

IA-32

```
                         Não há espaço a libertar
popl %ebx                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
j   $ra                  Regressa à função chamadora
```

`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 memória/*stack*: 14 no IA-32, 6 no MIPS !

IA-32

```
_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
```

Total: 63 *bytes*

ARM

```
_swap:
   str   fp, [sp, #-4]!
   add   fp, sp, #0      ; IA-32: mov sp, fp

   ldr   r3, [r0, #0]    ; IA-32: mov 0(r0), r3
   ldr   r2, [r1, #0]
   str   r2, [r0, #0]    ; IA-32: mov r2, 0(r0)
   str   r3, [r1, #0]

   add   sp, fp, #0
   pop   {fp}            ; pop é pseudo-instr
   bl    lr              ; branch & link
_call_swap:
   push  {fp, lr}        ; push é pseudo-instr
   add   fp, sp, #4
   sub   sp, sp, #8

   ldr   r3, .L3
   str   r3, [fp, #-12]
   ldr   r3, .L3+4
   str   r3, [fp, #-8]
   sub   r0, fp, #12     ; &zip1= fp+12
   sub   r1, fp, #8      ; &zip2= fp+8
   bl    _swap

   sub   sp, fp, #4
   pop   {fp, pc}        ; pop {pc} = ret
.L3:
   .word 15213
   .word 91125
```