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 *vs* RISC
6. Acesso e manipulação de dados estruturados

Relembrando: IA-32 versus Intel 64 (x86-64)

Principal diferença na organização interna:

- organização dos registos (na codificação de funções)
  - 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 **(8)** **&** para passagem e uso de argumentos **(6)**
- impacto na execução de funções:
  - menor utilização da *stack* na arquitetura Intel 64
  - Intel 64 potencialmente mais eficiente

Análise de um exemplo (`swap`) ...

x86-64: 64-bit extension to IA-32
Intel 64: Intel implementation of x86-64

Chamada de funções (IA-32 vs x86-64):

- 6 Registos para argumentos

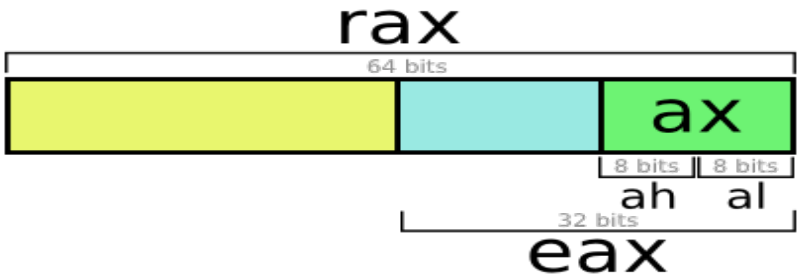
`%rdi`
`%rsi`
`%rdx`
`%rcx`
`%r8`
`%r9`

- Restantes argumentos: stack

Arg 7
Arg *8*
• • •
Arg n

- Valor de retorno

`%rax`



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

Revisão da codificação de `swap` *e* `call_swap` *no IA-32*

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

```
pushl %ebp
movl %esp, %ebp
subl $8, %esp

movl $15213, -4(%ebp)
movl $91125, -8(%ebp)

leal -8(%ebp), %eax
pushl %eax
leal -4(%ebp), %eax
pushl %eax
call swap
addl $8, %esp

leave
ret
```

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

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

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

movl -4(%ebp), %ebx
leave
ret
```

Funções em assembly: IA-32 versus Intel 64

x86-64 gcc 12.2 -02 **-m32** -fno-omit-frame-pointer

IA-32

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

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

        movl -4(%ebp), %ebx
        leave
        ret

_call_swap:
        pushl %ebp
        movl %esp, %ebp
        subl $8, %esp

        movl $15213, -4(%ebp)
        movl $91125, -8(%ebp)

        leal -8(%ebp), %eax
        pushl %eax
        leal -4(%ebp), %eax
        pushl %eax
        call swap

        leave
        ret
```

Intel 64

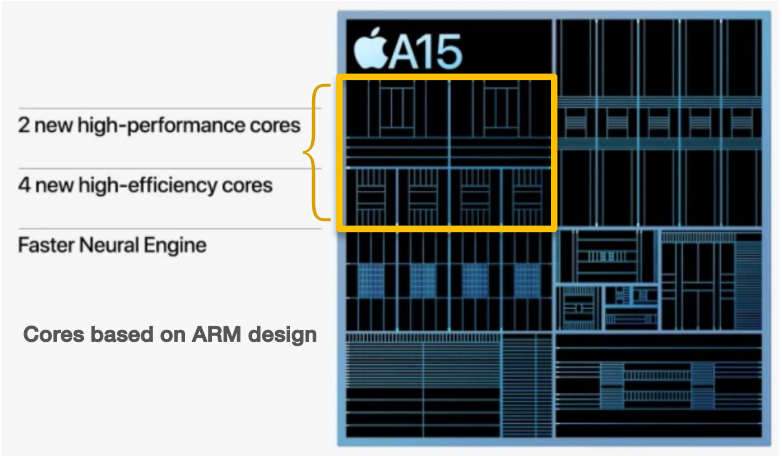
```
swap:
        pushl %rbp
        movl %rsp, %rbp

        movl (%rdi), %eax
        movl (%rsi), %edx
        movl %edx, (%rdi)
        movl %eax, (%rsi)


        leave
        ret


call_swap:
        pushq %rbp
        movq %rsp, %rbp
        subq $8, %rsp

        movl $15213, -4(%rbp)
        movl $91125, -8(%rbp)

        leaq -8(%rbp), %rsi
        leaq -4(%rbp), %rdi
        call swap

        leave
        ret
```

Total de acessos à *stack*:
15 no IA-32, 7 ~~(9)~~ no Intel 64 !

Caracterização das arquiteturas RISC

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

Arquiteturas RISC: em todos os *smartphones*!



Análise do nível ISA: o modelo RISC versus IA-32 (1)

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:
    $Mem[C^{te}+(Reg_b)]$ ou $Mem[(Reg_b)+(Reg_i)]$
  - ou poucos 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 em cada instrução
  - por ex. `push/pop` (IA-32)
    substituído pelo par de operações elementares
    `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

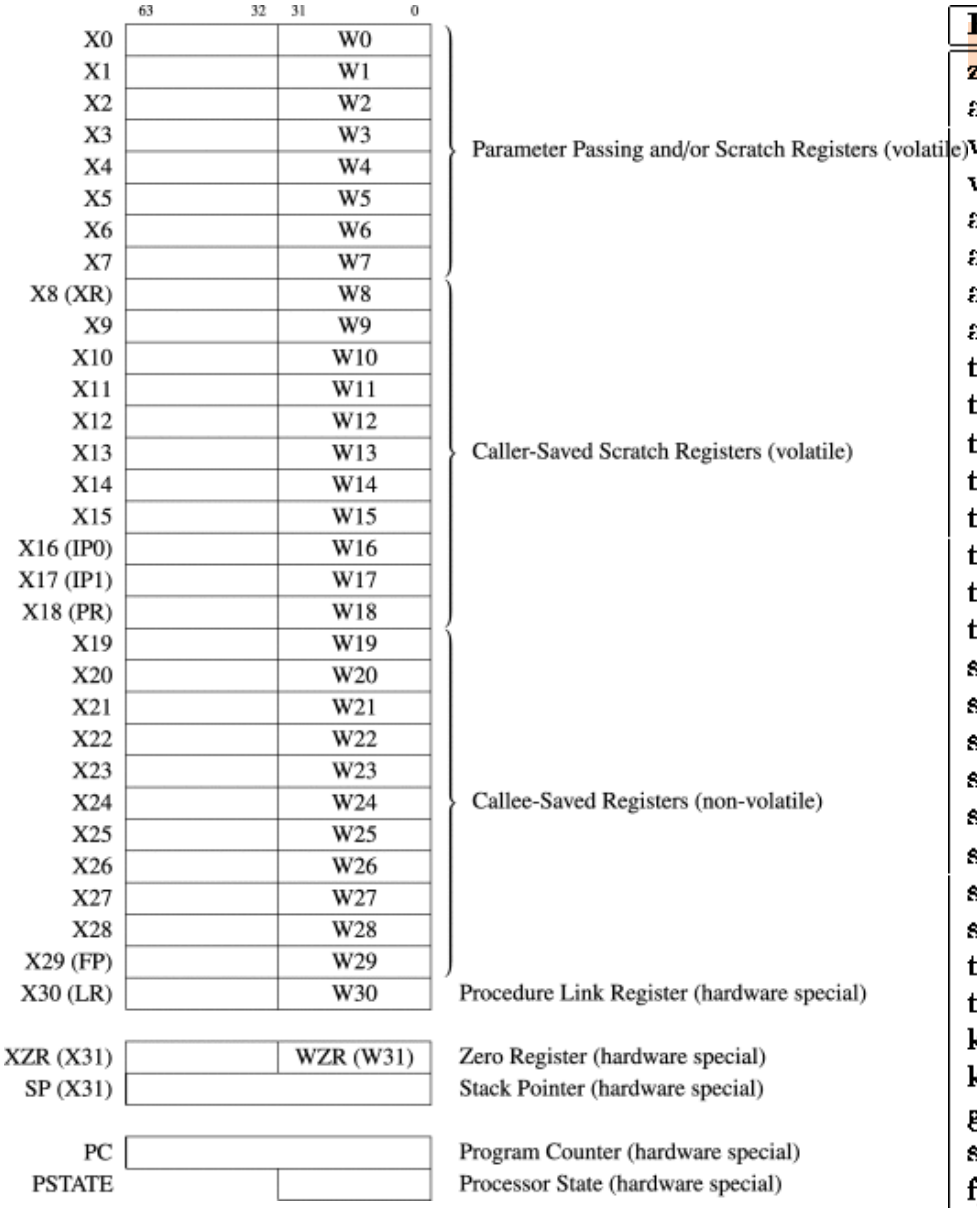
Name	Format	Example								Comments
Field size		4 bits	2 bits	1 bit	4 bits	1 bit	4 bits	4 bits	12 bits	All ARM instructions are 32 bits long
DP format	DP	Cond	F	I	Opcode	S	Rn	Rd	Operand2	Arithmetic instruction format
DT format	DT	Cond	F	Opcode			Rn	Rd	Offset12	Data transfer format
Field size		4 bits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

Principal diferença na organização interna:

- organização dos registos (na codificação de funções)
  - 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
- impacto na execução de funções:
  - menor utilização da *stack* nas arquiteturas RISC
  - RISC potencialmente mais eficiente

Análise de um exemplo (`swap`) ...

Convenção na utilização dos registos ARM/MIPS



Register Name	Number	Usage
zero	0	Constant 0
at	1	Reserved for assembler
v0	2	Expression evaluation and
v1	3	results of a function
a0	4	Argument 1
a1	5	Argument 2
a2	6	Argument 3
a3	7	Argument 4
t0	8	Temporary (not preserved across call)
t1	9	Temporary (not preserved across call)
t2	10	Temporary (not preserved across call)
t3	11	Temporary (not preserved across call)
t4	12	Temporary (not preserved across call)
t5	13	Temporary (not preserved across call)
t6	14	Temporary (not preserved across call)
t7	15	Temporary (not preserved across call)
s0	16	Saved temporary (preserved across call)
s1	17	Saved temporary (preserved across call)
s2	18	Saved temporary (preserved across call)
s3	19	Saved temporary (preserved across call)
s4	20	Saved temporary (preserved across call)
s5	21	Saved temporary (preserved across call)
s6	22	Saved temporary (preserved across call)
s7	23	Saved temporary (preserved across call)
t8	24	Temporary (not preserved across call)
t9	25	Temporary (not preserved across call)
k0	26	Reserved for OS kernel
k1	27	Reserved for OS kernel
gp	28	Pointer to global area
sp	29	Stack pointer
fp	30	Frame pointer
ra	31	Return address (used by function call)

Funções em assembly: IA-32 versus Intel 64

x86-64 gcc 12.2 -02 **-m32** ~~-fno-omit-frame-pointer~~

IA-32

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

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

     movl -4(%ebp), %ebx
     leave
     ret
_call_swap:
     pushl %ebp
     movl %esp, %ebp
     subl $8, %esp

     movl $15213, -4(%ebp)
     movl $91125, -8(%ebp)

     leal -8(%ebp), %eax
     pushl %eax
     leal -4(%ebp), %eax
     pushl %eax
     call swap

     leave
     ret
```

Intel 64

```
swap:
  ~~pushl %rbp~~
  ~~movl %rsp, %rbp~~

  movl (%rdi), %eax
  movl (%rsi), %edx
  movl %edx, (%rdi)
  movl %eax, (%rsi)


  ~~leave~~
  ret


call_swap:
  pushq %rbp
  movq %rsp, %rbp
  subq $8, %rsp


  movl $15213, -4(%rbp)
  movl $91125, -8(%rbp)

  leaq -8(%rbp), %rsi
  leaq -4(%rbp), %rdi
  call swap

  leave
  ret
```

MIPS64

```
swap:
     ~~daddiu $sp,$sp,-16~~
     ~~sd $fp,8($sp)~~
     ~~move $fp,$sp~~
     lw $2,0($4)
     lw $3,0($5)
     sw $3,0($4)
     sw $2,0($5)
     ~~move $sp,$fp~~
     ~~ld $fp,8($sp)~~
     jr $31   # $ra
     ~~daddiu $sp,$sp,16~~
call_swap:
   daddiu $sp,$sp,-48
   sd $31,40($sp)
   sd $fp,32($sp)
   move $fp,$sp

   li $2,15213 # 0x3b6d
   sw $2,0($fp)
   li $2,65536 # 0x10000
   ori $2,$2,0x63f5
   sw $2,4($fp)

   move $4,$fp # &zip1
   daddiu $5,$fp,4 # &zip2
   jal swap

   ...
```

Total de acessos à *stack*:
15 no IA-32, 7 ~~(9)~~ no Intel 64,
5+2 no MIPS64

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

`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

```
sd $31,40($sp)      Salvag. reg c/ endereço regresso
move $4,$fp         Calcula & coloca &zip1 no reg arg 0
daddiu $5,$fp,4     Calcula & coloca &zip2 no reg arg 1
jal   swap          Invoca swap
```

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

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*

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

2. Corpo de `swap` ... **`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) (5)

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
```

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

`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   $ra,40($sp)
```

Espaço a libertar na stack: NÃO
Recupera reg c/ ender regresso

Total de acessos à memória/*stack* (incl. inicialização de var's em mem):

13(+2) **no IA-32, 5**(+2) **no MIPS !**

Funções em assembly: IA-32 versus ARM64

x86-64 gcc 12.2 -02 **-m32** -fno-omit-frame-pointer

IA-32

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

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

     movl -4(%ebp), %ebx
     leave
     ret
_call_swap:
     pushl %ebp
     movl %esp, %ebp
     subl $8, %esp

     movl $15213, -4(%ebp)
     movl $91125, -8(%ebp)

     leal -8(%ebp), %eax
     pushl %eax
     leal -4(%ebp), %eax
     pushl %eax
     call swap

     leave
     ret
```

Intel 64

```
swap:
  pushl %rbp
  movl %rsp, %rbp

  movl (%rdi), %eax
  movl (%rsi), %edx
  movl %edx, (%rdi)
  movl %eax, (%rsi)


  leave
  ret

call_swap:
  pushq %rbp
  movq %rsp, %rbp
  subq $8, %rsp


  movl $15213, -4(%rbp)
  movl $91125, -8(%rbp)



  leaq -8(%rbp), %rsi
  leaq -4(%rbp), %rdi
  call swap

  leave
  ret
```

ARM64

```
swap:

     ldr w3, [x1] # w3 = M[x1]
     ldr w2, [x0] # w2 = M[x0]
     str w3, [x0] # M[x0] = w3
     str w2, [x1] # M[x1] = w2


     ret  # PC = x30 (link reg.)


call_swap:
   stp x29, x30, [sp, -32]!
   mov x29, sp  # x29 - fp ou bp


   mov w0, 15213
   str w0, [sp, 28] # M[sp+28]=w0
   mov w0, 25589
   movk w0, 0x1, lsl 16
   str w0, [sp, 24] # M[sp+24]=w0

   add x1, sp, 24
   add x0, sp, 28
   bl swap       # branch and link

   ldp x29, x30, [sp], 32
   ret
```

Nota:

```
stp x29, x30, [sp, -32]! # sp=sp-32; M[sp]=x29; M[sp+8]=x30;
```

Linguagens de programação que GCC suporta

- C com dialetos:
  - ANSI C original *(X3.159-1989, ISO standard ISO/IEC 9899:1990) aka* ***C89*** ou ***C90***; usar com '`-ansi`', '`-std=c90`'
  - ***C94*** ou ***C95***, usar com '`-std=iso9899:199409`'
  - ***C99***, usar com '`-std=c99`' or '`-std=iso9899:1999`'
  - ***C11***, usar com '`-std=c11`'
  - ***C17***, usar com '`-std=c17`'
  - por omissão GCC usa ***C11*** com extensões '`-std=gnu11`'
- C++ (usado com comando '`g++`') com dialetos:
  - ***C++98***, ***C++03***, ***C++11***, ***C++14***, ***C++17***, ***...***
- Fortran (usado com comando '`gfortran`') com dialetos:
  - ***Fortran 95***, ***Fortran 2003***, ***Fortran 2008***, ***...***

Processadores que GCC suporta

- especificado sempre com opção '`-m`'
- opção x86, usar com '`-march=`*cpu-type*'; algumas escolhas:
  - '***`native`***' (o sistema local)
  - '***`i386`***'... '***`x86-64`***'... '***`knl`***'... '***icelake***'... '***znver3***'
- opção MIPS, usar com '`-march=`*arch*'
  - é possível ainda selecionar um dado processador MIPS
- opção ARM, usar com '`-march=`*name*'
  - é possível ainda selecionar um dado processador ARM; mais comuns:
    - os da última geração de 32 bits, '***`armv7`***'
    - da 1ª geração de 64 bits, '***`armv8-a`***'