Comparación *primeras* RISC vs CISC

RISC (Ejemplo: ARM) ***Reduced Instruction Set Computer***	***CISC (Ejemplo: x86)*** ***Complex Instruction Set Computer***
Todas las instrucciones se codifican en 32 bits	Instrucciones de tamaño variable, algunas de decenas de bytes
Pocos formatos de instrucción, 3 operandos	Innumerables formatos de instrucción, 1, 2, 3, etc. operandos
Muchos registros, típicamente 31, todos de uso general	Pocos registros de uso general: 8 en x86 (16 en amd64)
Operandos en registros o valores inmediatos	Algunos operandos en memoria
Instrucciones load/store para transferir datos entre memoria y registros	También poseen instrucciones para transferir datos entre memoria y registros
Pequeño conjunto de instrucciones simples y generales	Gran conjunto de instrucciones complejas y especializadas
Implementación eficiente en pipeline	Implementación ineficiente
1 instrucción por ciclo del reloj típicamente	4 o más ciclos por cada instrucción
La mayoría de los transistores se ocupan para las operaciones	La mayoría de los transistores se ocupan para la unidad de control (microprogramada)
Diseñada para programar en lenguajes de alto nivel	Diseñadas para programar en assembler

Historia de los microprocesadores (computadores personales)

- Previo a 1978: Computadores *mainframe* de IBM y Digital
- 1978: Intel lanzó sus CPUs 8086, la última CISC, el set de instrucciones que iba a dominar el mundo

  registros y bus de datos de 16 bits, direcciones de 20 bits, 29K transistores
- 1979: Intel lanzó la versión económica, la 8088

  compatible con 8086, bus de datos de 8 bits
- 1981: **IBM** lanza el primer PC, basado en la 8088
- 1981: David Patterson publica ``RISC I: A Reduced Instruction Set VLSI Computer´´, 32 bits en ~40K transistores
- 1985: Intel lanza la 386, primera CPU de 32 bits con set de instrucciones x86, espacios de direcciones virtuales
- 1985: Mips, primera CPU RISC comercial
- Le suceden: Sparc de Sun Microsystems, ARM de Acorn Computers, PA-RISC de HP, RS6000 de **IBM**, Alpha 21064 de **Digital**
- *Nunca más se diseño una CPU CISC*

La sobrevivencia de la arquitectura x86

- 1989: Intel lanza la 486
- Se seleccionó un subconjunto de instrucciones simples y generales para ser implementadas eficientemente:
  - En pipeline, como las CPUs RISC
  - 1 instrucción por ciclo del reloj típicamente, como las RISC
  - Alta frecuencia de operación, como las RISC
  - Primera CPU con memoria cache en el chip para acelerar accesos a memoria, mejor que las RISC
- El restos de las instrucciones implementadas como en la 386 microprogramadas
  - Lentas: múltiples ciclos por instrucción
  - Instrucciones simples equivalentes se ejecutaban más rápido
- Los nuevos compiladores evitaban uso de instrucciones lentas
- Requería más transistores que una RISC pero se vendía 100 veces más que cualquier RISC
- Los mayores ingresos le han permitido a Intel fabricar CPUs CISC más rápidas que cualquier RISC, pero a costa de un mayor consumo

Innovaciones en la arquitectura de microprocesadores

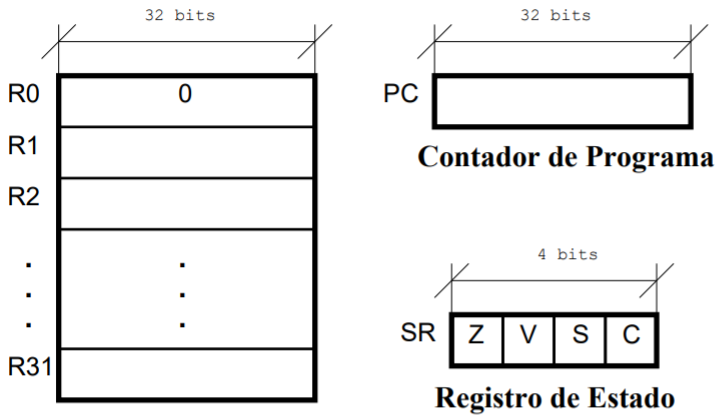
- 1993: Intel lanza el Pentium
  - Primer procesador superescalar: 2 instrucciones por ciclo del reloj
  - Más etapas en el pipeline: mayor frecuencia
- 1996: Intel lanza el Pentium Pro
  - Primer procesador con ejecución fuera de orden
  - Más etapas en el pipeline: mayor frecuencia
- 2003: Amd lanza Athlon 64, primer chip de 64 bits, arq. amd64
- 2005: Amd lanza Athlon 64 X2, primer chip dual core
- 2007: Apple lanza el iPhone
  - Primer smartphone con Unix: simplificación de MacOS
  - Procesador ARM fabricado por Samsung: RISC en pipeline de bajo consumo para smartphones
- 2009: iPhone 3GS, ARM A8, primer RISC superescalar
- 2010: Samsung lanza Galaxy S II, ARM A9, primer dual core, primer procesador con ejecución fuera de orden para smarphones
- 2020: Apple A13 Bionic amenaza la supremacía de Intel en cuanto a velocidad single thread

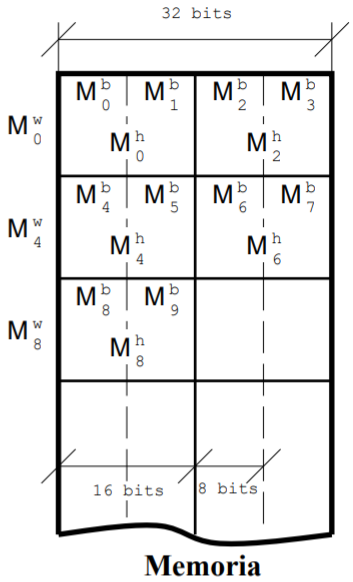
Arquitectura lógica de M32

- M32 es una CPU RISC inventada con fines pedagógicos para este curso
- RISC significa Reduced Instruction Set Computer
- No existen implementaciones reales de M32
- M32 está basada en la primera CPU RISC diseñada por Patterson a comienzos de los 80s
- La RISC de Patterson también fue la base de las CPUs Sparc de Sun Microsystems (90s y 2000s)
- Los objetivos de usar M32 en este curso son:
  - Aprender los principios básicos de las CPUs RISC
  - Aprender a implementar en circuitos una CPU simple
- Big endian

M32: Arquitectura Lógica

M32 es una CPU diseñada sólo con fines docentes y por lo tanto no posee todas las capacidades exigidas a una CPU real. La arquitectura de registros de M32 es la siguiente:





Dedicados:

R1 : return value
R1, R2, R3, ...:
parámetros
R29: frame pointer
R30: stack pointer
R31: return address

Nótese que una misma dirección en memoria puede ser vista como un *byte*, como parte de una media palabra o como parte de una palabra completa.

Las instrucciones *assembler* de M32 son de la forma:

Operador	Operandos	Descripción
`op`	reg_s, *val*, reg_d	operaciones del tipo $reg_d = reg_s$ `op` *val*
`op`	*addr*, *reg*	lectura en memoria
`op`	*reg*, *addr*	escritura en memoria
`salto`	*disp*	saltos condicionales

En donde cada uno de los operandos corresponde a:

reg	Cualquier registro entre $\%R_0$, $\%R_1$ ... $\%R_{31}$
imm	Un valor binario $\in [-2^{12}, 2^{12}-1]$ (representable en 13 bits)
addr	Una dirección de la forma $[reg' + reg'']$ o $[reg + imm]$
val	Un valor de la forma *reg* o *imm*
disp	Un desplazamiento $\in [-2^{23}, 2^{23}-1]$ (representable en 24 bits)

Definiciones	
$Rep_s^n(x)$	Representación en n bits con signo del entero x
$Rep_u^n(x)$	Representación en n bits sin signo del entero positivo x
Ext_0(x)	Conversión de x a 32 bits extendiendo con ceros
Ext_s(x)	Conversión de x a 32 bits extendiendo con el signo
$Trunc_b$(x)	Trunca la palabra x a 8 bits
$Trunc_h$(x)	Trunca la palabra x a 16 bits

Instrucciones aritméticas		
Instrucción assembler		Operación
`add`	$reg_s,\ val,\ reg_d$	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg_d \rangle_{t+1} = \langle reg_s \rangle_t \oplus \langle val \rangle_t$ $\langle v \rangle_{t+1} = Ovf(\langle reg_s \rangle_t, \langle val \rangle_t, 0)$ $\langle c \rangle_{t+1} = Carry(\langle reg_s \rangle_t, \langle val \rangle_t, 0)$
`addx`	$reg_s,\ val,\ reg_d$	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg_d \rangle_{t+1} = \langle reg_s \rangle_t \oplus \langle val \rangle_t \oplus \langle c \rangle_t$ $\langle v \rangle_{t+1} = Ovf(\langle reg_s \rangle_t, \langle val \rangle_t, \langle c \rangle_t)$ $\langle c \rangle_{t+1} = Carry(\langle reg_s \rangle_t, \langle val \rangle_t, \langle c \rangle_t)$
`sub`	$reg_s,\ val,\ reg_d$	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg_d \rangle_{t+1} = \langle reg_s \rangle_t \oplus \sim \langle val \rangle_t \oplus 1$ $\langle v \rangle_{t+1} = Ovf(\langle reg_s \rangle_t, \sim\langle val \rangle_t, 1)$ $\langle c \rangle_{t+1} = Carry(\langle reg_s \rangle_t, \sim\langle val \rangle_t, 1)$
`subx`	$reg_s,\ val,\ reg_d$	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg_d \rangle_{t+1} = \langle reg_s \rangle_t \oplus \sim \langle val \rangle_t \oplus \langle c \rangle_t$ $\langle v \rangle_{t+1} = Ovf(\langle reg_s \rangle_t, \sim\langle val \rangle_t, \langle c \rangle_t)$ $\langle c \rangle_{t+1} = Carry(\langle reg_s \rangle_t, \sim\langle val \rangle_t, \langle c \rangle_t)$
Instrucciones lógicas		
`and`	$reg_s,\ val,\ reg_d$	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle \%\mathtt{R}_d \rangle_{t+1} = \langle reg_s \rangle_t \& \langle val \rangle_t$
`or`	$reg_s,\ val,\ reg_d$	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle \%\mathtt{R}_d \rangle_{t+1} = \langle reg_s \rangle_t \mid \langle val \rangle_t$
`xor`	$reg_s,\ val,\ reg_d$	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle \%\mathtt{R}_d \rangle_{t+1} = \langle reg_s \rangle_t \mathtt{xor} \langle val \rangle_t$
Instrucciones de desplazamiento		
`sll`	$reg_s,\ val,\ reg_d$	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg_d \rangle_{t+1} = \langle reg_s \rangle_t << \langle val \rangle_t$
`srl`	$reg_s,\ val,\ reg_d$	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg_d \rangle_{t+1} = \langle reg_s \rangle_t >> \langle val \rangle_t$
`sra`	$reg_s,\ val,\ reg_d$	$reg_d \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg_d \rangle_{t+1} = \langle reg_s \rangle_t >>_s \langle val \rangle_t$

Para todas las instrucciones aritméticas se cumple:

$$\langle z \rangle_{t+1} = \begin{cases} 1 & \text{Si} \langle reg_d \rangle_{t+1} = 0 \\ 0 & \text{sino} \end{cases}$$

$\langle v \rangle_{t+1} = 0$ (a menos que se indique otra cosa)

$\langle s \rangle_{t+1} = Sign(\langle reg_d \rangle_{t+1})$

$\langle c \rangle_{t+1} = \langle c \rangle_t$ (a menos que se indique otra cosa)

Ejemplo

```
int exps(int i, int *a, int d) {
  return (-a[i]>>d)*10;
}
```

```
# i: R1, a: R2, d: R3
sll R1, 2, R4    # i*sizeof(int)
add R2, R4, R5   # a+i*sizeof(int)
ldw [R5], R5     # a[i]
sub 0, R5, R5    # -a[i]
sra R5, R3, R6   # e= -a[i] >> d
                 # No hay mult.
sll R6, 2, R7    # e * 4
add R6, R7, R8   # e * 5
add R8, R8, R1   # e * 10
jmpl R31, R0     # return
```

La función $\langle x \rangle_t$ que aparece en la segunda columna se define como:

$$\langle x \rangle_t = \textit{el valor del registro/memoria } x \textit{ en } t$$
$$\langle [reg' + reg''] \rangle_t = \langle reg' \rangle_t \oplus \langle reg'' \rangle_t$$
$$\langle imm \rangle_t = Ext_s(Rep_s^{13}(imm))$$
$$\langle [reg + imm] \rangle_t = \langle reg \rangle_t \oplus Ext_s(Rep_s^{13}(imm))$$

A continuación se describirá cada una de las posibles instrucciones assembler de M32.

Ejemplos:

ldw [R29+8], R5
Ssw R3, [R10+R11]
ldsb [R30+5]. R8

Instrucciones de lectura en memoria			
Instrucción assembler		Operación	Tipo leido
`ldw`	*addr*, *reg*	$\langle addr \rangle_t \& 3 \neq 0 \Rightarrow$ `TRAP` $reg \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg \rangle_{t+1} = \mathtt{M}^w_{\langle addr \rangle_t}$	una palabra
`lduh`	*addr*, *reg*	$\langle addr \rangle_t \& 1 \neq 0 \Rightarrow$ `TRAP` $reg \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg \rangle_{t+1} = Ext_0(\langle \mathtt{M}^h_{\langle addr \rangle_t} \rangle_t)$	media palabra sin signo
`ldub`	*addr*, *reg*	$reg \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg \rangle_{t+1} = Ext_0(\langle \mathtt{M}^b_{\langle addr \rangle_t} \rangle_t)$	*byte* sin signo
`ldsh`	*addr*, *reg*	$\langle addr \rangle_t \& 1 \neq 0 \Rightarrow$ `TRAP` $reg \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg \rangle_{t+1} = Ext_s(\langle \mathtt{M}^h_{\langle addr \rangle_t} \rangle_t)$	media palabra con signo
`ldsb`	*addr*, *reg*	$reg \not\equiv \%\mathtt{R}_0 \Rightarrow \langle reg \rangle_{t+1} = Ext_s(\langle \mathtt{M}^b_{\langle addr \rangle_t} \rangle_t)$	*byte* con signo
Instrucciones de escritura en memoria			
Instrucción assembler		Operación	Tipo escrito
`stw`	*reg*, *addr*	$\langle addr \rangle_t \& 3 \neq 0 \Rightarrow$ `TRAP` $\langle \mathtt{M}^w_{\langle addr \rangle_t} \rangle_{t+1} = \langle reg \rangle_t$	una palabra
`sth`	*reg*, *addr*	$\langle addr \rangle_t \& 1 \neq 0 \Rightarrow$ `TRAP` $\langle \mathtt{M}^h_{\langle addr \rangle_t} \rangle_{t+1} = Trunc_h(\langle reg \rangle_t)$	media palabra
`stb`	*reg*, *addr*	$\langle \mathtt{M}^b_{\langle addr \rangle_t} \rangle_{t+1} = Trunc_b(\langle reg \rangle_t)$	*byte*

Instrucciones de Salto	
Instrucción assembler	Operación
`b`*cond* *disp*	*cond* se verifica $\Rightarrow \langle PC \rangle_{t+1} = \langle PC \rangle_t \oplus Ext_s(Rep_s^{22}(disp)) \oplus 4$ *cond* no se verifica $\Rightarrow \langle PC \rangle_{t+1} = \langle PC \rangle_t \oplus 4$
`jmpl` *addr*,*reg*	$\langle PC \rangle_{t+1} = \langle addr \rangle_t$ $reg \not\equiv$ %$\mathtt{R}_0 \Rightarrow \langle reg \rangle_{t+1} = \langle PC \rangle_t \oplus 4$

Condiciones de Saltos		
Código Instrucción	Tipo de salto	Condición de salto
ba	incondicional	1
be	igualdad	z
bne	desigualdad	$\overline{z}$
Comparaciones con signo		
bg	mayor	$\overline{z}(sv + \overline{s}\ \overline{v})$
bge	mayor o igual	$sv + \overline{s}\ \overline{v}$
bl	menor	$s\overline{v} + \overline{s}v$
ble	menor o igual	$s\overline{v} + \overline{s}v + z$
Comparaciones sin signo		
bgu	mayor	$c\overline{z}$
bgeu	mayor o igual	c
blu	menor	$\overline{c}$
bleu	menor o igual	$\overline{c} + z$

Ejemplo: min(a, b)

```
    # a: R1, b: R2
    sub R1, R2, R0
    bl Es_a
    or R2, 0, R3
    ba cont
Es_a:
    or R1, 0, R3
Cont:
```