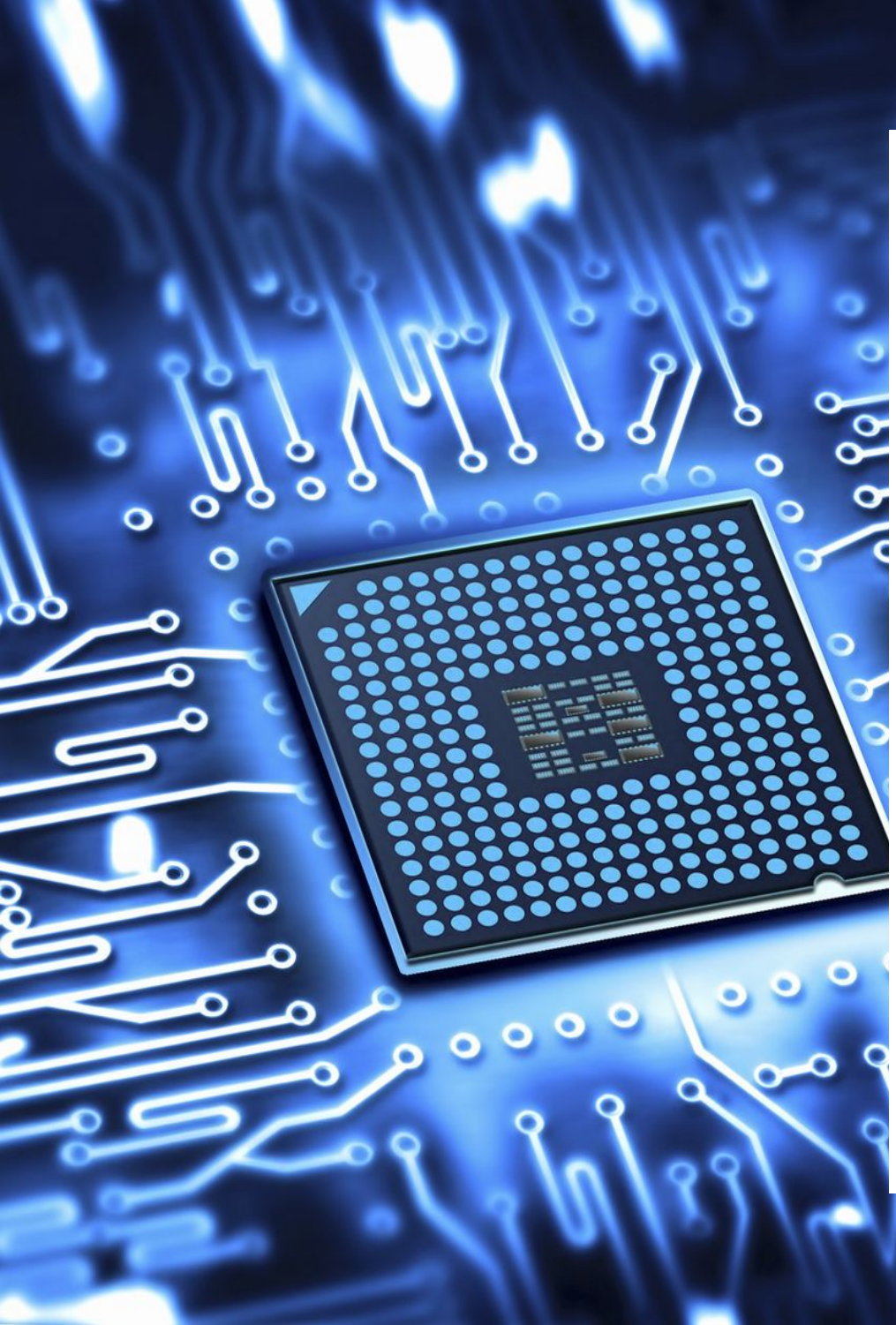


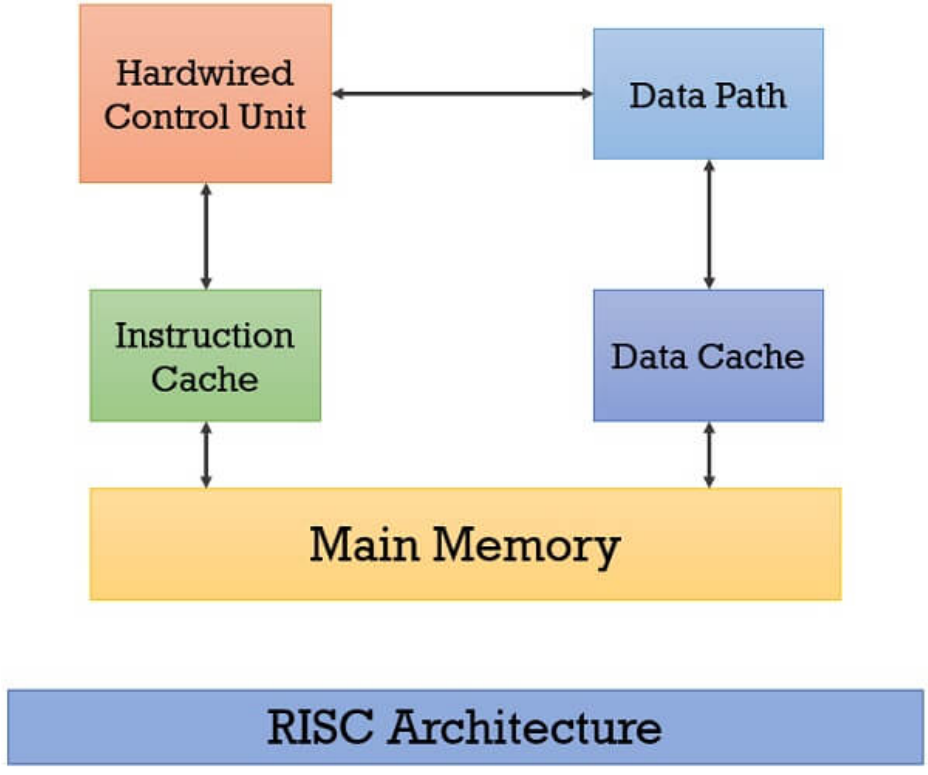
Arquitectura de computadoras

Instrucciones

Universidad Nacional de la Patagonia

RISC vs CISC

CISC	RISC
Instrucciones multiciclo	Instrucciones de único ciclo
Carga/almacenamiento incorporadas en otras instrucciones	Carga/almacenamiento son instrucciones separadas
Arquitectura memoria-memoria	Arquitectura registro-registro
Instrucciones largas, Código con menos líneas	Instrucciones cortas, Código con más líneas
Utiliza memoria de microprograma	Implementa las instrucciones directamente en hardware
Se enfatiza la versatilidad del repertorio de instrucciones	Se añaden instrucciones nuevas sólo si son de uso frecuente y no reducen el rendimiento de las más importantes
Reduce la dificultad de implementar compiladores	Compiladores complejos
	Elimina microcódigo y la decodificación de instrucciones complejas

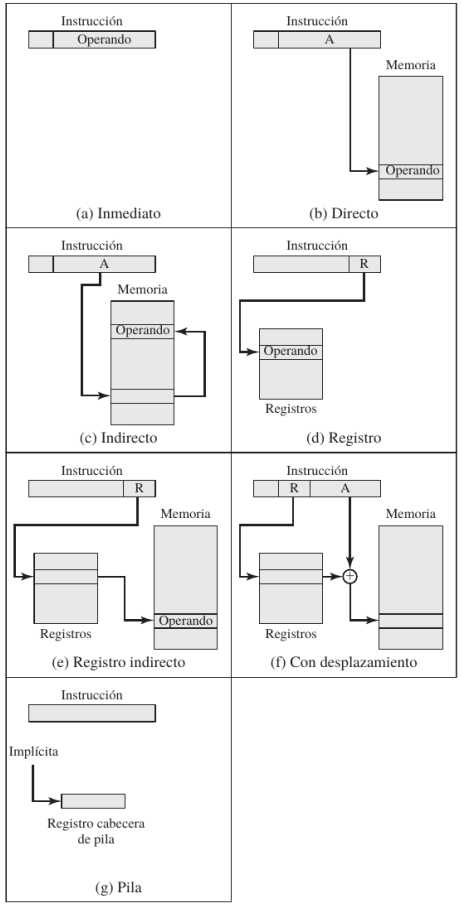


- Ventajas de RISC
  - Las instrucciones máquina son más simples
  - Poseen una cantidad muy reducida de instrucciones
  - La unidad de control es cableada para una ejecución más veloz
  - Las instrucciones poseen modos de direccionamiento sencillos
  - La mayoría de las instrucciones operan usando solo registros y evitan el acceso a memoria
  - Resulta sencillo implementar segmentación
  - Se ejecuta una instrucción por ciclo de reloj
- Desventajas de RISC
  - Mayor cantidad de instrucciones que el mismo programa para una arquitectura CISC
  - Unidad de control cableada, por lo que las modificaciones son costosas
  - Requiere mucho trabajo realizar operaciones complejas e implementar modos de direccionamiento complejos
  - No se permite el acceso directo a memoria

RISC-V

Modos de direccionamiento de memoria

- Inmediato
- Directo
- Indirecto
- Registro
- Indirecto con registro
- Con desplazamiento (load y stores)
- Pila
- Relativo al PC (saltos condicionales)



Formatos de instrucciones

Consideraciones para especificar un formato de instrucciones:

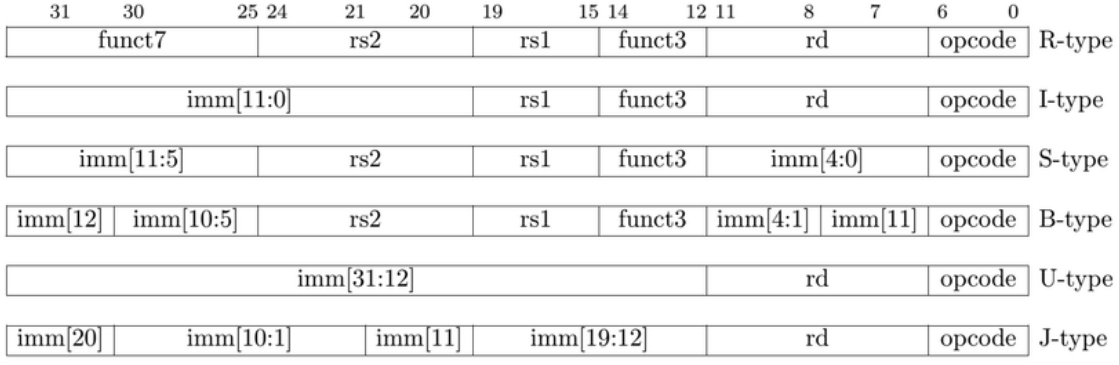
- Número de modos de direccionamiento
- Número de operandos
- Registros frente a memoria
- Número de conjuntos de registros
- Rango de direcciones
- Granularidad de las direcciones

Formatos de instrucciones RISC-V (RV32I)

- Todas las instrucciones de RISC-V ocupan 32 bits (4 bytes)
- Contenido de las instrucciones:
  - opcode (código de operación)
  - Operandos. Valores o ubicación para reliazar la operación en:
    - Registros
    - Números inmediatos (constantes)
    - Etiquetas (direcciones de otras lineas del programa)

Formatos de instrucciones RISC-V

- Tipo R: para operaciones entre registros
- Tipo I: para inmediatos cortos y traer de memoria a un registro (load)
- Tipo S: para almacenar en memoria (store)
- Tipo B: para saltos condicionales (branches)
- Tipo U: para inmediatos largos
- Tipo J: para saltos incondionales



Instrucciones tipo R, tipo I y tipo U

- Operación binaria (dos operandos) que puede aplicarse a dos registros
  - Utiliza el formato R

```
add t0, t1, t2  # t0 <- t1 + t2
```

- Operación binaria que puede aplicarse a un registro y a un valor inmediato (constante)
  - Utiliza el formato I

```
addi t0, t1, 123    # t0 <- t1 + 123
```

- Operación cuyo valor inmediato requiere mayor cantidad de bits
  - Utiliza el formato U

```
lui t1, 100000      # t1[31:12] <- 100000
```

- Si queremos cargar constantes de 32 bits? Por ejemplo: 0xAAAAA123
- Debemos usar dos instrucciones: una de ellas es "load upper immediate"

```
lui t1, 0xAAAAA
```

10101010101010101010	000000000000

- A continuación, colocamos los bits en el lugar correcto

```
ori t1, t1, 0x123
```

10101010101010101010	000000000000
00000000000000000000	000100100011

ori

10101010101010101010	000100100011

Instrucciones tipo I y tipo S

- Operación de carga (se trae dato de memoria principal a un registro)
  - Utiliza el formato I

```
# el registro t1 posee la dirección de memoria de una variable
lw t4, 2(t1)    # t4 <- MP[t1 + 2]
```

- Operación de almacenamiento (guarda el dato de un registro en memoria principal)
  - Utiliza el formato S

```
# el registro t2 posee la dirección de memoria de una variable
sw t3, 0(t2)    # MP[t2 + 0] <- t3
```

Instrucciones tipo B y tipo J

- Operación de salto condicional (solo salta si se cumple una condición)
  - Utiliza el formato B

```
# finloop es una etiqueta (representa a una dirección de memoria de
    interés)
beq t3, t4, finloop    # if (t3 == t4) then PC <- finloop
```

- Operación de salto incondicional (siempre salta)
  - Utiliza el formato J

```
# rutina es una etiqueta
jal t1, rutina    # t1 <- PC; PC <- rutina
```

Pseudoinstrucciones

	Pseudo-instrucción	Instrucciones
Asignar valor inmediato a registro	`li t1, 3`	`addi t1, zero, 3`
Asignar dirección de memoria a registro	`la t1, N`	`auipc t1, 0x0fc10` `addi t1, t1, 0x008`
Salto incondicional	`j destino`	`jal zero, destino`
Salto condicional	`bgtz t1, destino`	`blt zero, t1, destino`

Extensiones

- Instrucción addi formato tipo I
  - Posee 12 bits disponibles para un valor inmediato
  - Los registros son de 32 bits de ancho
- No podemos sumar un valor de 12-bit a un valor de 32-bit
  - Las palabras deben ser de la misma anchura
  - Debemos **convertir** el valor inmediato de 12-bit a uno equivalente de 32-bit
  - Convertimos **extendiendo** 20-bit el valor
- La extensión requiere que cualquier valor de forma estrecha deba ser convertido a una forma más ancha

Extensión de cero

- Los valores sin signo pueden ser extendidos agregándoles ceros adelante

42 000000101010

42 0000000000000000000 000000101010

Extensión de signo

- La extensión de cero no funciona para valores negativos

  -42 111111010110

  4054 000000000000000000 111111010110

- Para números negativos tipo signed, extendemos el bit de signo

  -42 111111010110

  -42 11111111111111111111 111111010110

Instrucciones sin signo

- Algunas instrucciones tiene versiones unsigned (sin signo)
  - Especiales para cálculos con valores sin signo
  - RISC-V trata a los registros que contiene estos tipos de valores como:
    - Para algunas operaciones, las versiones unsigned ignoran el overflow
    - Para algunas operaciones,las versiones unsigned se extienden a cero en lugar del signo
  - La mayoría de las instrucciones unsigned se le agrega u al opcode

Tamaños de los datos

- RISC-V trabaja con 3 tamaños de datos: word (32 bits), half (16 bits) y byte (8 bits)
- Las instrucciones de carga y almacenamiento tienen variantes para cada tamaño de datos
- A los datos de tamaño half y byte las instrucciones lx y sx realizan extensión de signo

Registros

Registro	Nombre	Descripción	Pres.
x0	zero	Siempre cero	
x1	ra	Dirección de retorno	No
x2	sp	Puntero a pila	Si
x3	gp	Puntero global	
x4	tp	Puntero a hilo	
x5	t0	Registro de enlace temporal	No
x6-7	t1-2	Temporales	No
x8	s0/fp	Registro a guardar / puntero de marco	Si
x9	s1	Registro a guardar	Si
x10-11	a0-1	Argumentos de rutina / valores de retorno	No
x12-17	a2-7	Argumentos de rutina	No
x18-27	s2-11	Registros a guardar	Si
x28-31	t3-6	Temporales	No

Registros (para números de coma flotante)

Registro	Nombre	Descripción	Pres.
f0-7	ft0-7	Temporales	No
f8-9	fs0-1	Registros a guardar	Si
f10-11	fa0-1	Argumentos de rutina / valores de retorno	No
f12-17	fa2-7	Argumentos de rutina	No
f18-27	fs2-11	Registros a guardar	Si
f28-31	ft8-11	Temporales	No

```
# implementa un programa en ensamblador que defina un arreglo de enteros V, inicializado
# a los siguientes valores V = [1,-4, -5,-2], y obtenga como resultado una variable
# booleana RES, que será 1 si todos los elementos de este arreglo son menores que cero.

        .data                   # inicio de la seccion de datos
V:      .word -1, 4, -5, -2     # elementos del vector V
N:      .word 4                 # cantidad de elementos del vector V
RES:    .space 4                # reservo 4 bytes de espacio

        .text                   # inicio de la seccion de instrucciones
main:
        # guardamos las direcciones de memoria
        # en los registros deseados
        la s1, V                # s1 <- &V
        la t1, N                # t1 <- &N
        la t2, RES              # t2 <- &RES

        # inicializamos registros
        li t3, 0                # i <- 0
        lw t4, 0(t1)            # t4 <- N
        li t5, 4                # t5 <- tamaño de cada elemento en bytes

loop:   beq t3, t4, finloop
        mul t6, t3, t5          # se calcula desplazamiento
        add t6, t6, s1          # se calula direccion efectiva

        lw s0, 0(t6)            # s0 <- V[i]

        bgtz s0, es_mayor       # si s0 es positivo, salta a es_mayor
        addi t3, t3, 1          # i <- i + 1
        j loop
```

```
loop:   beq t3, t4, finloop
        mul t6, t3, t5          # se calcula desplazamiento
        add t6, t6, s1          # se calula direccion efectiva

        lw s0, 0(t6)            # s0 <- V[i]

        bgtz s0, es_mayor       # si s0 es positivo, salta a es_mayor
        addi t3, t3, 1          # i <- i + 1
        j loop

es_mayor:
        sw zero, 0(t2)          # RES <- 0
        j finprograma

finloop:
        li t3, 1
        sw t3, 0(t2)            # RES <- 1

finprograma:
        li a7, 10               # indico que quiero terminar el programa
        ecall                   # termino el programa
```