Procesadores de lenguajes. Práctica 1.

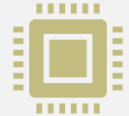
CARACTERÍSTICAS GENERALES DEL LENGUAJE TINTO

Objetivos

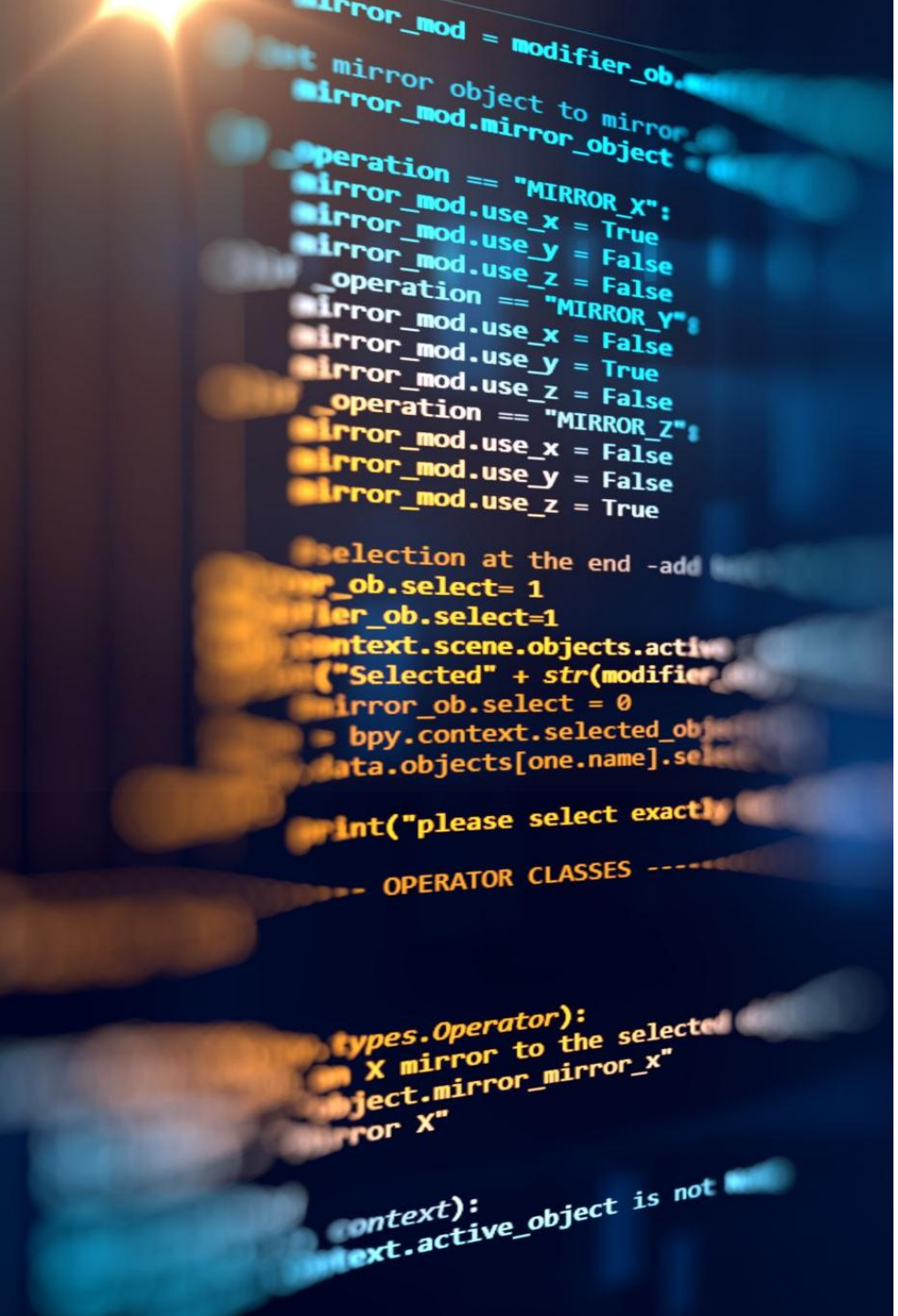
1. Presentar el lenguaje de programación *Tinto*

2. Describir el proceso de compilación y ejecución de programas en *Tinto*

3. Presentar el procesador MIPS



4. Presentar el simulador QT-SPIM



1. Presentar el lenguaje de programación *Tinto*

- Tinto es un lenguaje de programación imperativo orientado a procesos (tipo C)
- Lenguaje de programación muy simple.
- Para los aspectos de diseño de *Tinto* (especificación léxica, sintáctica y semántica, tipos de datos, instrucciones, etc.) se ha optado por simplificación del lenguaje para desarrollar un compilador tanto de forma manual como con la ayuda de herramientas.

1. Presentar el lenguaje de programación *Tinto*

Ejemplo fichero fuente descrito en *Tinto*

```
import Math;
library Ejemplo {
  public int MaximoComunDivisor(int a, int b) {
    int mcd = Math.min(a,b);
    while(mcd>0) {
       if( a%mcd == 0 && b%mcd == 0) return mcd;
       mcd = mcd - 1;
    }
    return 1;
  }
}
```

1. Presentar el lenguaje de programación *Tinto*

Características de *Tinto*:

- Palabras reservadas: 15 palabras reservadas. C = 32, C++ = 50, Java = 53, etc.
  - *boolean, char, else, false, if, import, int, library, native, private, public, return, true, void, y while.*
- Tipos de datos: tres tipos de datos: *int, char* y *boolean*. Todos son almacenados en registros de 32 bits.
  - *char*: código ASCII de 8 bits. Desde el menos significativo del 0 al 7 (ASCII) del 8 a al 31 todo cero.
  - *boolean*: 0 para false y 1 para true.
  - *void*: la función no devuelve nada.

1. Presentar el lenguaje de programación *Tinto*

Características de *Tinto*:

❿Instrucciones: Funciones de Tinto en lista de instrucciones.

- Declaraciones de variables locales (se admite asignación en declaración y declaración conjunta de variables del mismo tipo).
- Instrucciones de asignación simple (por medio del operador =)
- Instrucciones de ejecución de una función
- Instrucciones condicionales (if-else)
- Bucles (while)
- Finalización de las funciones

1. Presentar el lenguaje de programación *Tinto*

Características de *Tinto*:

- Operadores aritméticos y lógicos.
  - Operadores aritméticos:
    - Suma (+), resta (-), multiplicación (*), división (/), módulo(%)
  - Operadores lógicos:
    - AND (&&), OR (||) y NOT (!)

1. Presentar el lenguaje de programación *Tinto*

Características de *Tinto*:

❿ Operadores relaciones: para comparaciones entre los valores de distintas expresiones:

- *int* permite: mayor (>), menor (<), mayor o igual (>=) y menor o igual (<=)
- para todos (*char, boolean, int*): igual (==) y distinto (!=)

1. Presentar el lenguaje de programación *Tinto*

Características de *Tinto*:

❿Literales: para cada tipo de datos:

- *int* (decimal, octal, hexadecimal y binario)
- *char* (entre comillas simples: caracteres imprimibles, caracteres de escape y caracteres en formato octal)
- *boolean* (true o false)

1. Presentar el lenguaje de programación *Tinto*

Características de *Tinto*:

❿Componentes del lenguaje:

- Ficheros descritos en Tinto definen bibliotecas de funciones.
- Cada fichero contiene una lista de bibliotecas importadas (*import*) y una biblioteca (*library*) que coincide con el nombre del fichero.
- El Código de las funciones puede llamar a otras funciones públicas o privadas de la misma biblioteca.
- Llamadas a funciones públicas de otras bibliotecas. (nombre_biblioteca.nombre_funcion)
- Para utilizar alguna biblioteca esta ha tenido que ser importada.

1. Presentar el lenguaje de programación *Tinto*

Características de *Tinto*:

❿Componentes del lenguaje:

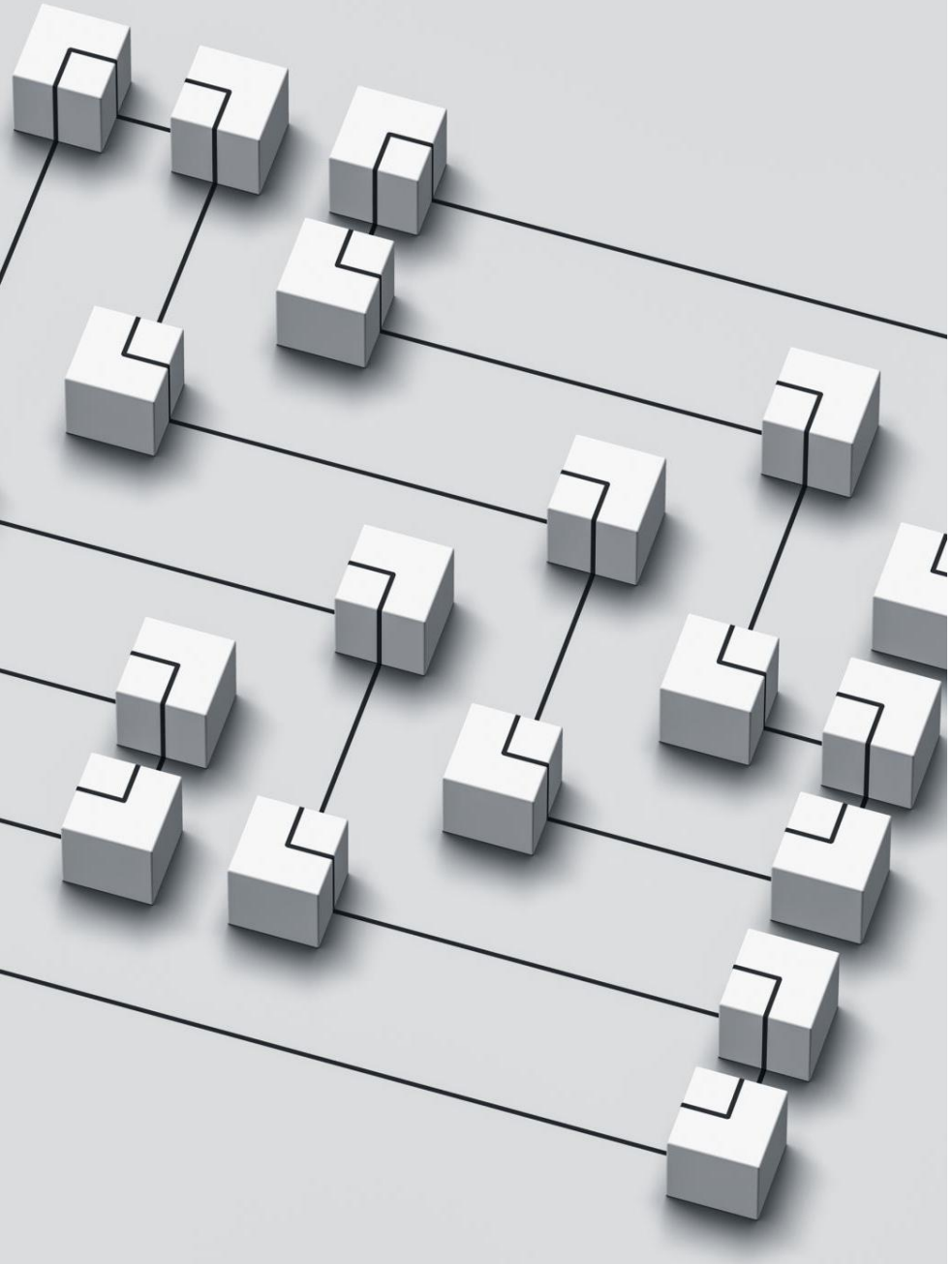
- También se puede definir bibliotecas nativas (*native*) que están formadas por una lista de declaraciones de funciones. El Código está directamente en ensamblador y no se describe en el fichero Fuente. Con estas bibliotecas nativas se pueden utilizar funciones que usan instrucciones de ensamblador específicas que no genera el compilador, como llamadas al sistema (con syscall).

2. Describir el proceso de compilación y ejecución de programas en *Tinto*

- El compilador de Tinto se encuentra en el archivo tintoc.jar.
- El objetivo del compilador es compilar el archivo Main.tinto, que debe contener la función Main() que marca el comienzo de la aplicación programada.
- A partir de esta biblioteca se analizan y compilan el resto de bibliotecas importadas.
- El directorio por defecto es donde se ejecuta el comando de compilación, pero se puede modificar con Path.
- Las bibliotecas deben encontrarse en el directorio de trabajo o usar –I
- El resultado de la compilación es un archivo con la extensión “.s”, se modifica -o

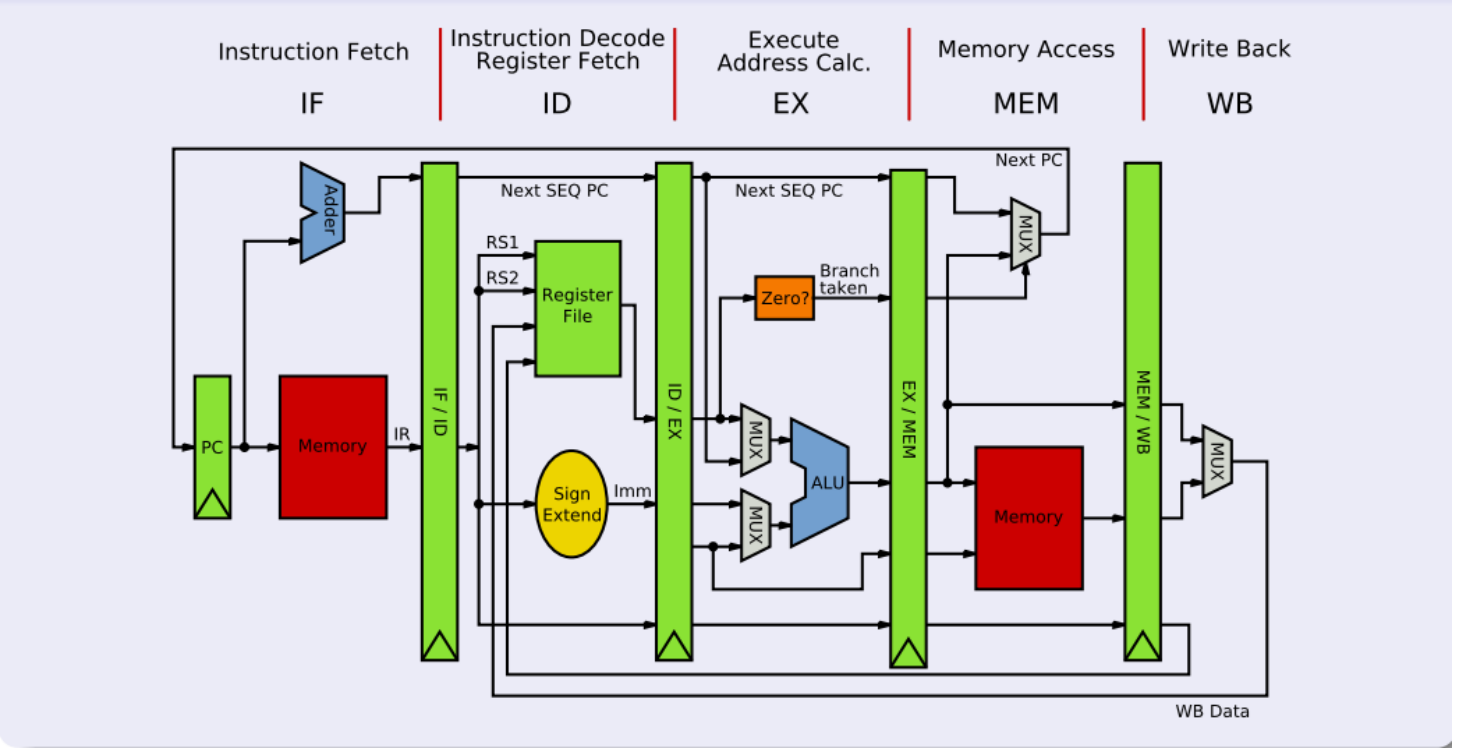
2. Describir el proceso de compilación y ejecución de programas en *Tinto*

- Descarga y descompresión de la Práctica 1: https://uhu.es/francisco.moreno/gii_pl/practicas/practica01/practica01.rar
- Abrimos una terminal en una subcarpeta de la carpeta examples, por ejemplo: ./examples/1. Hola Mundo y ejecutamos el siguiente comando:
  - java -jar ../tintoc.jar -I ../Utils
- Generara un archivo Application.s o en caso de error TintocErrors.txt con el error.
- Con –v se puede generar el código intermedio. Archivo Main.tic
- Resultado de la compilación es un fichero en ensamblador del procesador MIPS. Después lo probaremos en QT-SPIM.



3. Presentar el procesador MIPS

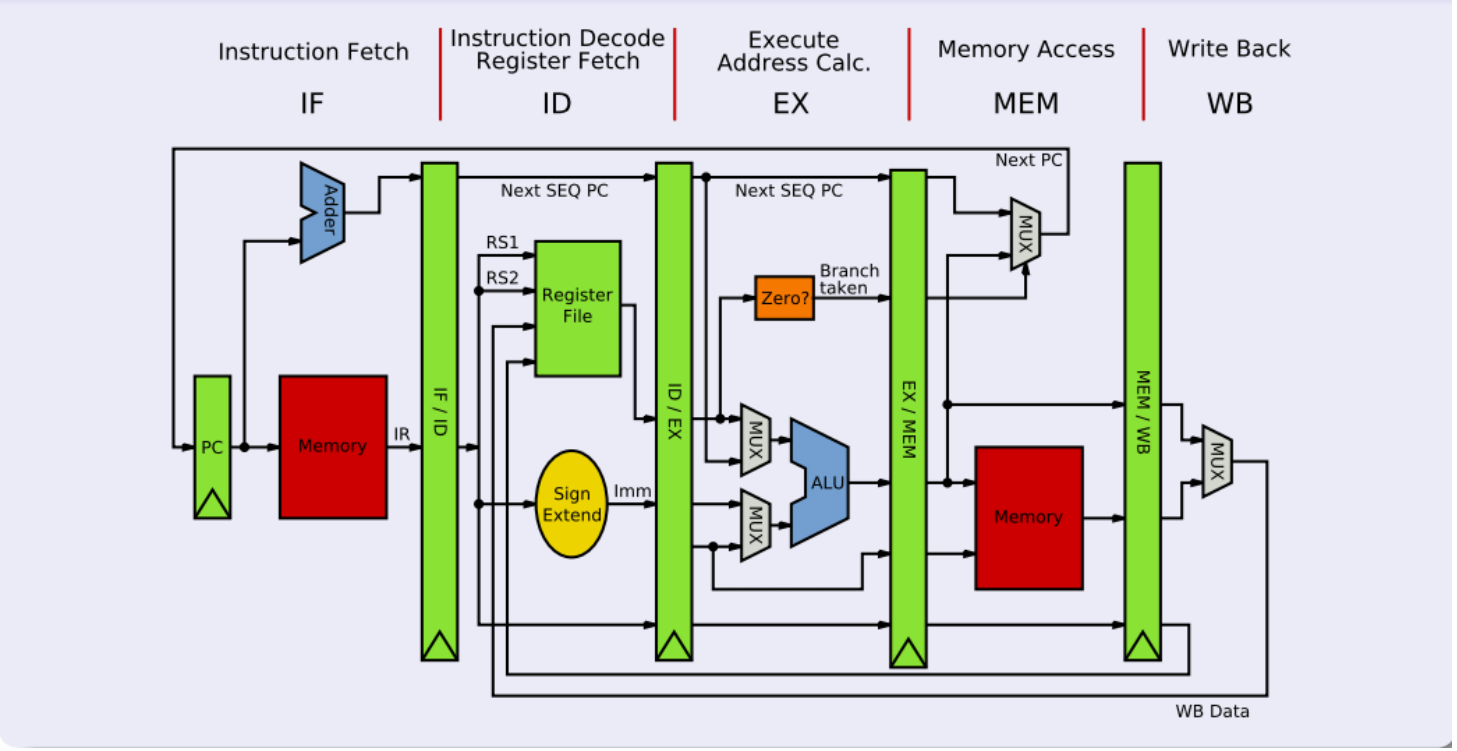
- MIPS (Microprocessor without Interlocked Pipeline Stages) es el nombre de una familia de procesadores RISC desarrollados por MIPS Technologies.
- RISC (Reduced Instruction Set Computing):
  - Conjunto de instrucciones reducido
  - Ejecución rápida y eficiente
  - Arquitectura de carga/almacenamiento (solo carga/almacenamiento operan en la memoria, el resto en registros)
  - Múltiples registros
  - Pipeline optimizado (ejecución en paralelo de instrucciones)
  - Formato de instrucción fijo (todas mismo tamaño)
  - Menos modos de direccionamiento (reduce complejidad acceso memoria)



3. Presentar el procesador MIPS

- MIPS Microarquitectura Etapas:

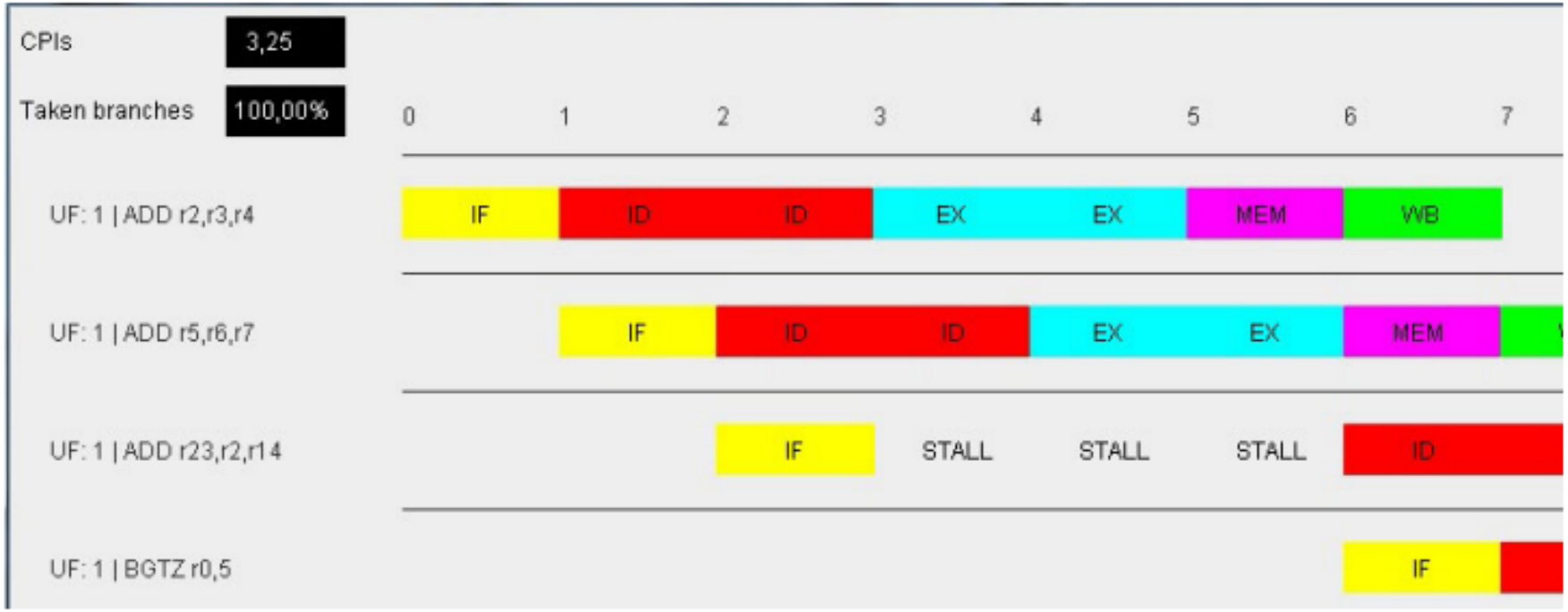
1. Instruction Feth: instrucción desde memoria de instrucciones, actualiza contador de programa y usa un multiplexor para la siguiente instrucción
2. Instrucción decode & Register Fetch: decodificación de la instrucción, lectura de registros necesarios y se extiende el símbolo de los operandos



3. Presentar el procesador MIPS

- MIPS Microarquitectura Etapas:

3. Execute & Address Calculation: operación aritmética/lógica en la ALU. Determina si hay rama condicional Branch Taken. Multiplexores para seleccionar los operandos.
4. Memory Access: En caso de operaciones de carga/almacenamiento se accede a memoria.
5. Write Back: resultado de la ALU o de la memoria en registros



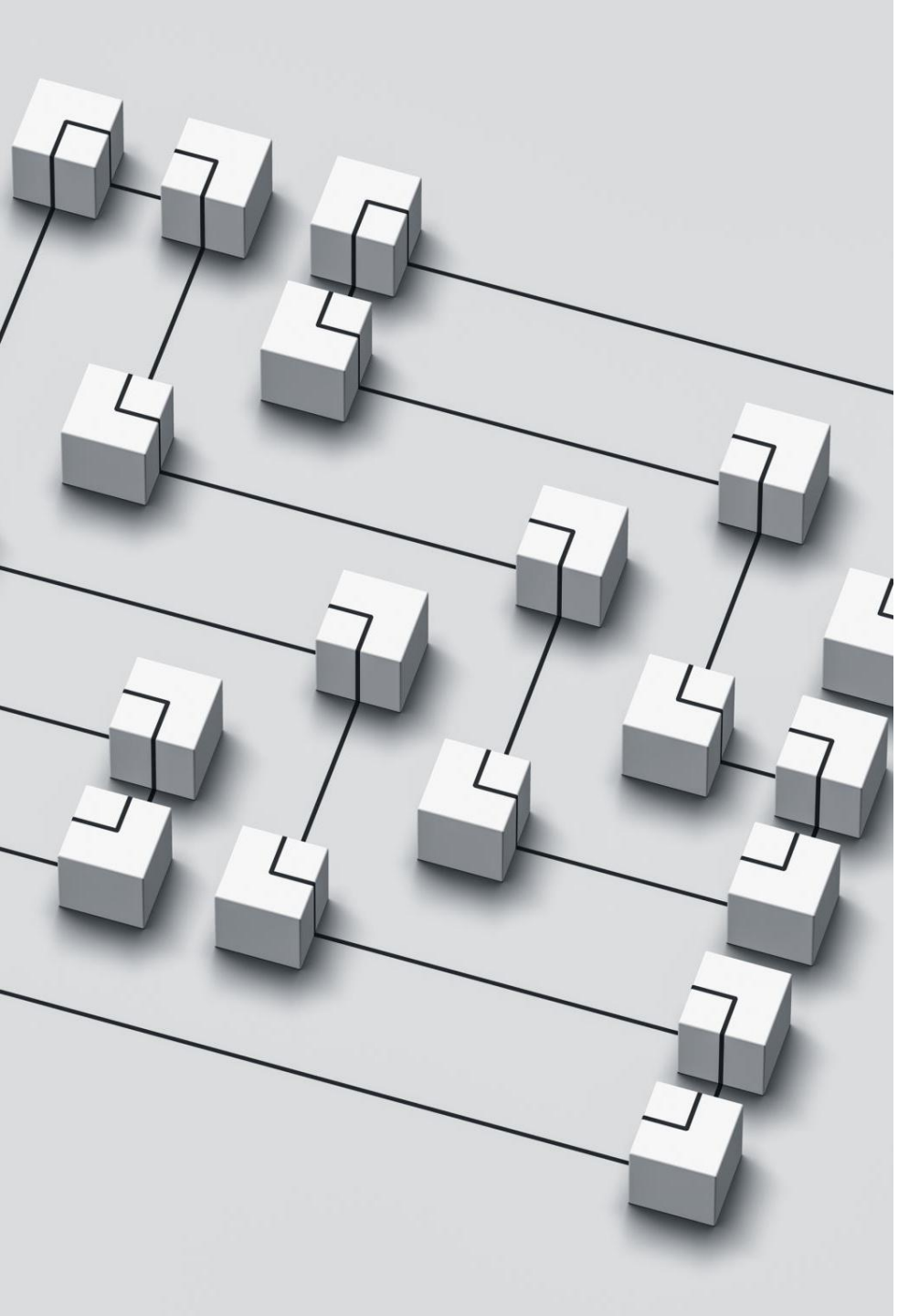
3. Presentar el procesador MIPS

MIPS PIPELINE

ABS.D	ABS.PS[1]	ABS.S	ADD	ADD.D	ADD.PS[1]	ADD.S	ADDI
ADDIU	ADDU	ALNV.PS[1]	AND	ANDI	BC1F	BC1FL	BC1T
BC1TL	BC2F	BC2FL	BC2T	BC2TL	BEQ	BEQL	BGEZ
BGEZAL	BGEZALL	BGEZL	BGTZ	BGTZL	BLEZ	BLEZL	BLTZ
BLTZAL	BLTZALL	BLTZL	BNE	BNEL	BREAK	C.cond.D	C.cond.PS[1]
C.cond.S	CACHE	CEIL.L.D[1]	CEIL.L.S[1]	CEIL.W.D	CEIL.W.S	CFC1	CFC2
CLO	CLZ	COP2	CTC1	CTC2	CVT.D.L[1]	CVT.D.S	CVT.D.W
CVT.L.D[1]	CVT.L.S[1]	CVT.PS.S[1]	CVT.S.D	CVT.S.L[1]	CVT.S.PL[1]	CVT.S.PU[1]	CVT.S.W
CVT.W.D	CVT.W.S	DERET	DI[2]	DIV	DIV.D	DIV.S	DIVU
EHB[2]	EI[2]	ERET	EXT[2]	FLOOR.L.D[1]	FLOOR.L.S[1]	FLOOR.W.D	FLOOR.W.S
INS[2]	J	JAL	JALR	JALR.HB[2]	JR	JR.HB[2]	LB
LBU	LDC1	LDC2	LDXC1[1]	LH	LHU	LL	LUI
LUXC1[1]	LW	LWC1	LWC2	LWL	LWR	LWXC1[1]	MADD
MADD.D[1]	MADD.PS[1]	MADD.S[1]	MADDU	MFC0	MFC1	MFC2	MFHC1[2]
MFHC2[2]	MFHI	MFLO	MOV.D	MOV.PS[1]	MOV.S	MOVF	MOVF.D
MOVF.PS[1]	MOVF.S	MOVN	MOVN.D	MOVN.PS[1]	MOVN.S	MOVT	MOVT.D
MOVT.PS[1]	MOVT.S	MOVZ	MOVZ.D	MOVZ.PS[1]	MOVZ.S	MSUB	MSUB.D[1]
MSUB.PS[1]	MSUB.S[1]	MSUBU	MTC0	MTC1	MTC2	MTHC1[2]	MTHC2[2]
MTHI	MTLO	MUL	MUL.D	MUL.PS[1]	MUL.S	MULT	MULTU
NEG.D	NEG.PS[1]	NEG.S	NMADD.D[1]	NMADD.PS[1]	NMADD.S[1]	NMSUB.D[1]	NMSUB.PS[1]
NMSUB.S[1]	NOR	OR	ORI	PLL.PS[1]	PLU.PS[1]	PREF	PREFX[1]
PUL.PS[1]	PUU.PS[1]	RDHWR[2]	RDPGPR[2]	RECIP.D[1]	RECIP.S[1]	ROTR[2]	ROTRV[2]
ROUND.L.D[1]	ROUND.L.S[1]	ROUND.W.D	ROUND.W.S	RSQRT.D[1]	RSQRT.S[1]	SB	SC
SDBBP	SDC1	SDC2	SDXC1[1]	SEB[2]	SEH[2]	SH	SLL
SLLV	SLT	SLTI	SLTIU	SLTU	SQRT.D	SQRT.S	SRA
SRAV	SRL	SRLV	SSNOP	SUB	SUB.D	SUB.PS[1]	SUB.S
SUBU	SUXC1[1]	SW	SWC1	SWC2	SWL	SWR	SWXC1[1]
SYNC	SYNCI[2]	SYSCALL	TEQ	TEQI	TGE	TGEI	TGEIU
TGEU	TLBP	TLBR	TLBWI	TLBWR	TLT	TLTI	TLTIU
TLTU	TNE	TNEI	TRUNC.L.D[1]	TRUNC.L.S[1]	TRUNC.W.D	TRUNC.W.S	WAIT
WRPGPR[2]	WSBH[2]	XOR	XORI				

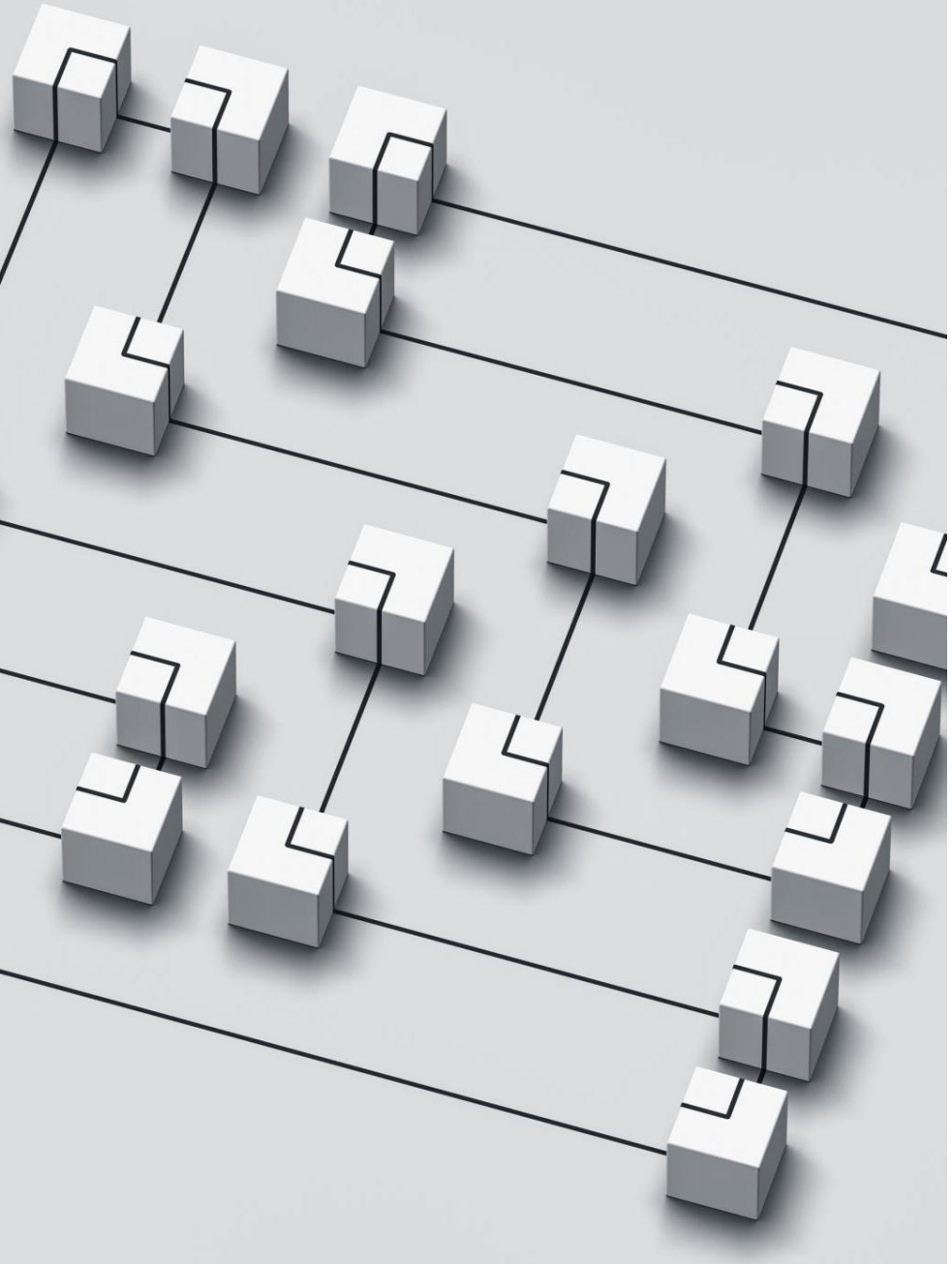
3. Presentar el procesador MIPS

MIPS SET INSTRUCTION:



3. Presentar el procesador MIPS

- Documentación:
  - Arquitectura: https://uhu.es/francisco.moreno/gii_pl/practicas/practica01/MD00082-2B-MIPS32INT-AFP-02.50.pdf
  - Conjunto de Instrucciones: https://uhu.es/francisco.moreno/gii_pl/practicas/practica01/MD00086-2B-MIPS32BIS-AFP-02.50.pdf
  - Arquitectura de recursos privilegiados: https://uhu.es/francisco.moreno/gii_pl/practicas/practica01/MD00090-2B-MIPS32PRA-AFP-02.50.pdf



3. Presentar el procesador MIPS

- Características:
  - Arquitectura 32/64 bits.
  - Unidad en coma flotante que admite float de 32 bits y double de 64 bits.
  - Simulación del código por medio de QT-SPIM
  - Cuenta con una unidad de propósito general que tiene 32 registros de 32 bits (del $0 al $31) y una unidad en coma flotante con 32 registros de 32 bits ($f0 a $f31) que se pueden utilizar como 16 registros de 64 bits para almacenar en formato double.
  - Los registros de propósito general pueden usarse libremente en las instrucciones, el registro $0 está cableado y que el registro $31 almacena la dirección de retorno cuando se ejecuta la instrucción JAL. Hay un convenio de uso de los registros .

$0	$zero	Cableado a cero	$16	$s0	Información a salvar
$1	$at	Reservado para el Ensamblador	$17	$s1	Información a salvar
$2	$v0	Valor devuelto por una función	$18	$s2	Información a salvar
$3	$v1	Valor devuelto por una función	$19	$s3	Información a salvar
$4	$a0	Paso de argumentos	$20	$s4	Información a salvar
$5	$a1	Paso de argumentos	$21	$s5	Información a salvar
$6	$a2	Paso de argumentos	$22	$s6	Información a salvar
$7	$a3	Paso de argumentos	$23	$s7	Información a salvar
$8	$t0	Registro temporal	$24	$t8	Registro temporal
$9	$t1	Registro temporal	$25	$t9	Registro temporal
$10	$t2	Registro temporal	$26	$k0	Reservado para el Kernel
$11	$t3	Registro temporal	$27	$k1	Reservado para el Kernel
$12	$t4	Registro temporal	$28	$gp	Global pointer
$13	$t5	Registro temporal	$29	$sp	Stack pointer
$14	$t6	Registro temporal	$31	$fp	Frame pointer
$15	$t7	Registro temporal	$31	$ra	Return address

3. Presentar el procesador MIPS

CONVENIO DE USO DE LOS REGISTROS MIPS

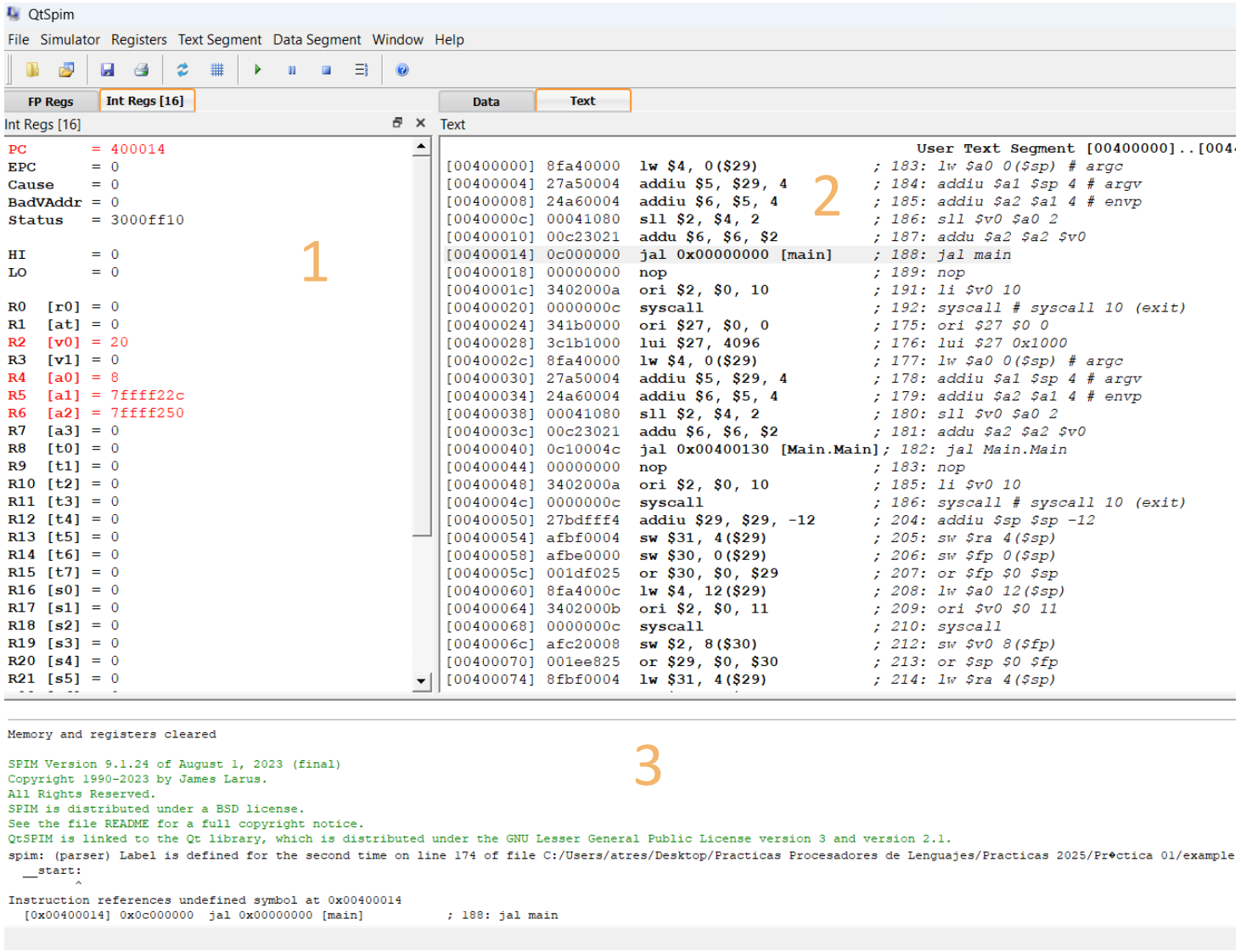
4. Presentar el simulador QT-SPIM

- SPIM es un simulador que ejecuta programas descritos en ensamblador de MIPS32. Programada por James Larus distribuida libremente.
- Versión más reciente QT-SPIM
- Enlace de descarga: https://sourceforge.net/projects/spimsimulator/files/
- El simulador desarrolla un conjunto mínimo de llamadas al sistema que permiten acceder a una consola y acceder a ficheros.

4. Presentar el simulador QT-SPIM

TABLA DE LLAMADAS AL SISTEMA

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $a0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	



4. Presentar el simulador QT-SPIM

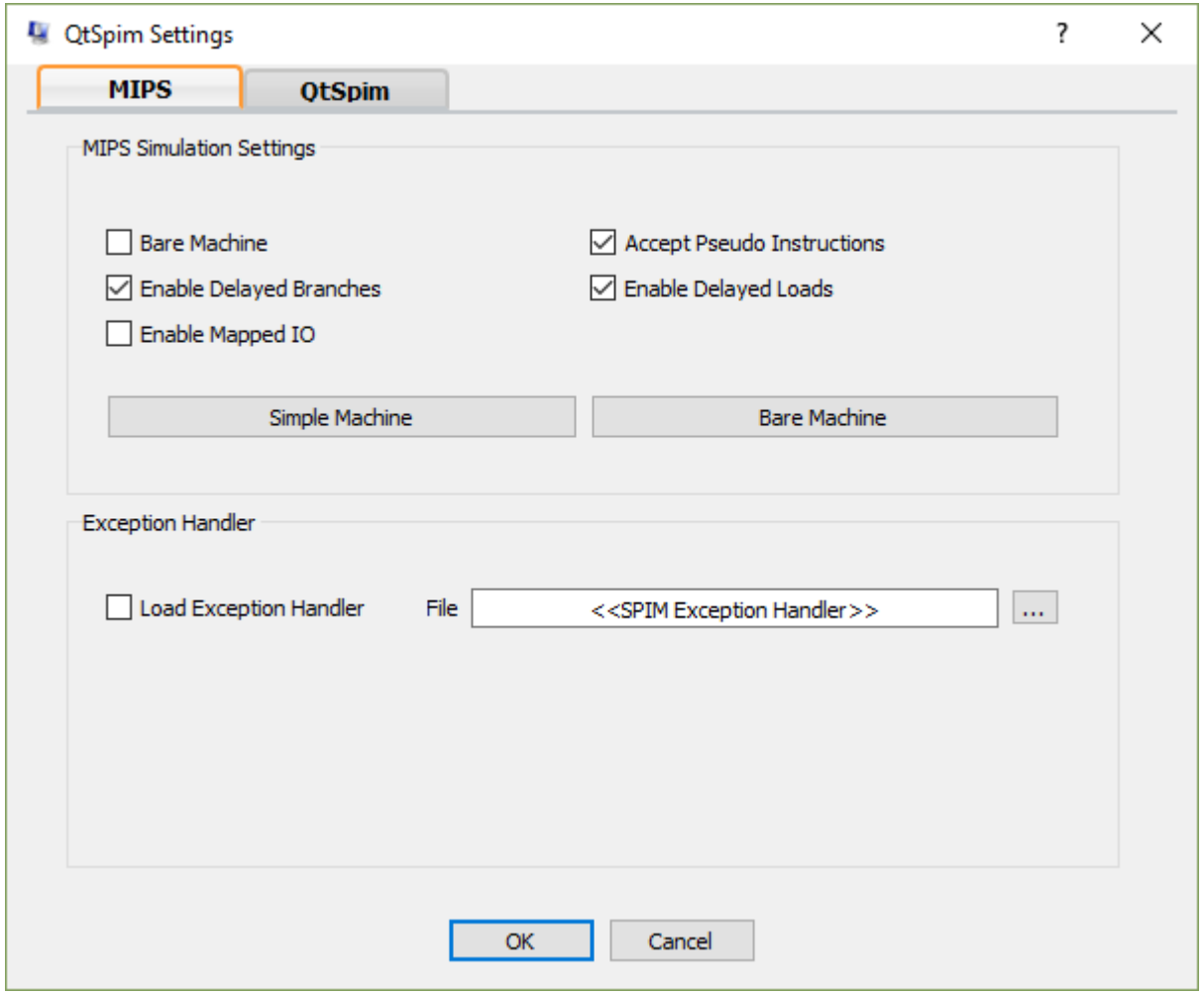
QT-SPIM:

1. Registros del procesador
   1. Registros de propósito general (GPR)
   2. Registros de la unidad de coma flotante (FPR)
2. Contenido de la memoria
   1. Segmento de texto (instrucciones del programa)
   2. Segmento de datos (pila, memoria estática y dinámica)
3. Muestra mensajes

```
290         nop
291         .end    Console.readChar
292
293
294     #-----------------------------------------------------
295     # Copyright (c) 2017, Francisco JosÃ© Moreno Velo
296     # All rights reserved.
297     #-----------------------------------------------------
298     #-----------------------------------------------------
299     # Main.Main
300     #-----------------------------------------------------
301     main:
302         .globl  Main.Main
303         .ent    Main.Main
304     Main.Main:
305         addiu $sp $sp 0xffffff9c
306         sw $ra 92($sp)
307         sw $fp 88($sp)
308         or $fp $0 $sp
309         addiu $sp $sp 0xfffffffc
310         ori $a0 $0 0x48
311         sw $a0 0($fp)
312         nop
313         lw $a0 0($fp)
```

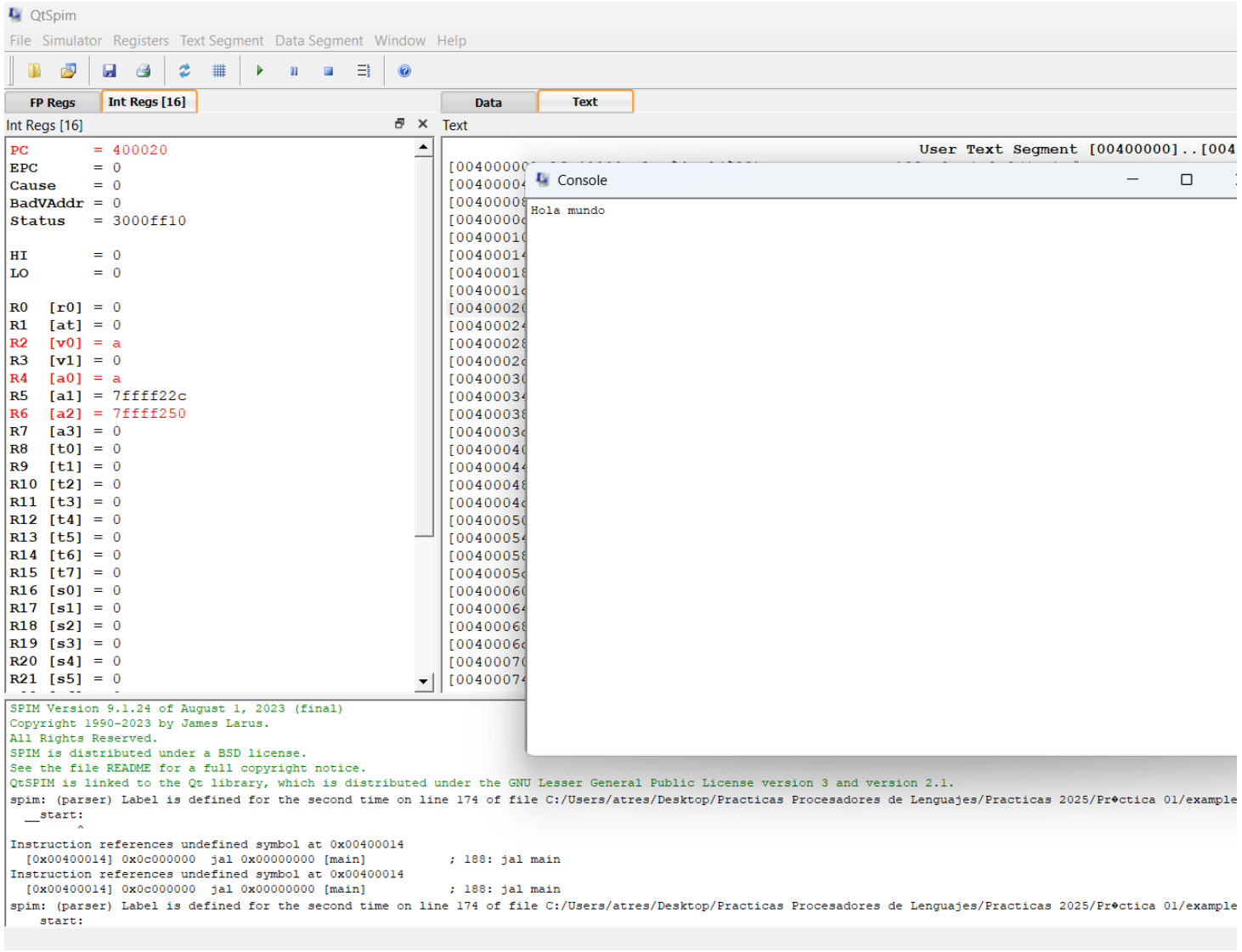
4. Presentar el simulador QT-SPIM

- En los archivos “.s” hay que incluir “main: “
- Se carga con File->Load File
- Se puede ejecutar completamente con el triangulo verde o con F5
- Y paso a paso con el icono de la lista step a step o con F10



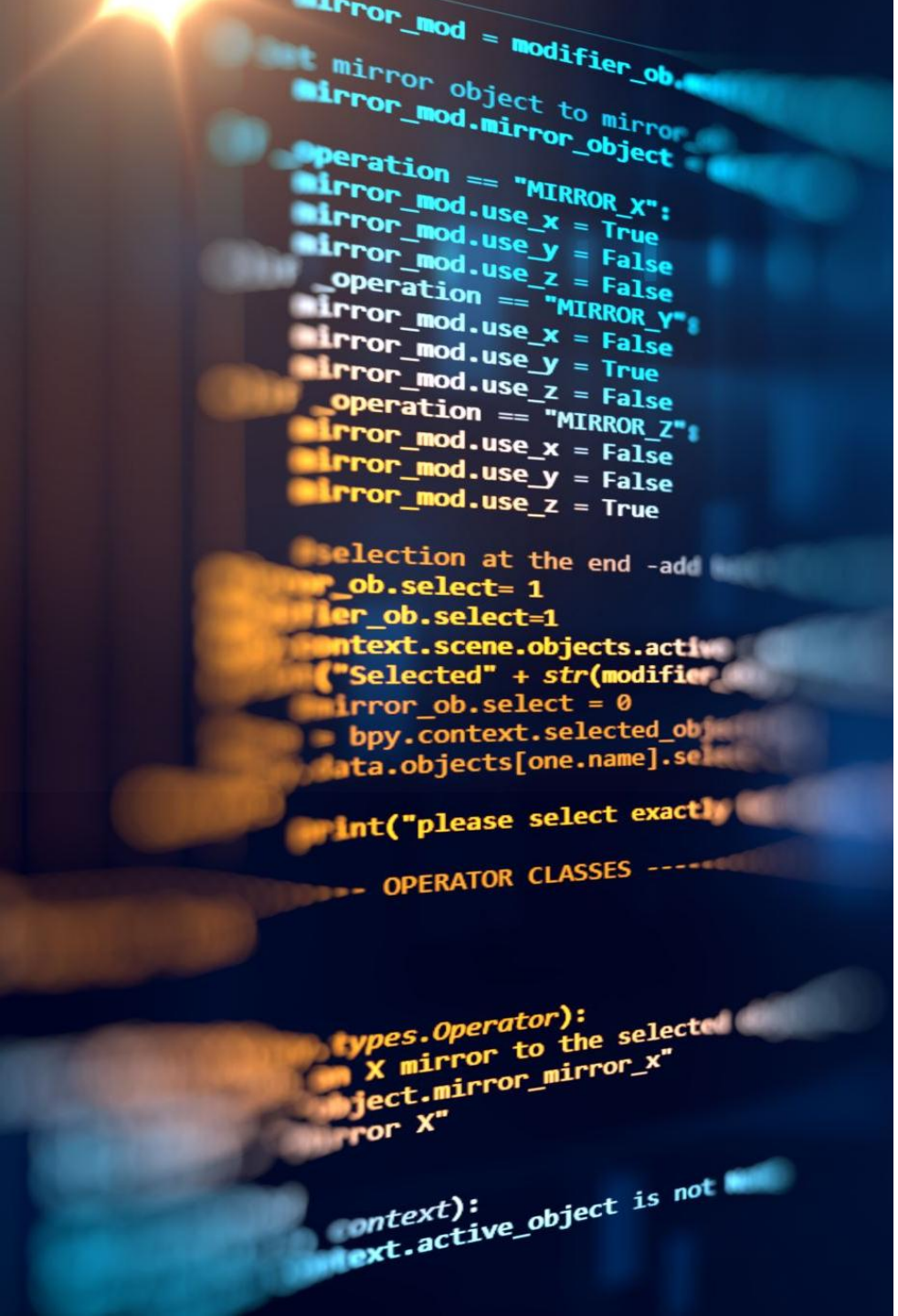
4. Presentar el simulador QT-SPIM

- Simulador->Settings
- Opciones que deben estar habilitadas:
  - "Enable Delayed Branches"
  - "Enable Delaye Loads"
  - "Accept Pseudo Instructions"
- El compilador de Tinto utiliza algunas pseudo-instrucciones
- "Load Exception Handler" deshabilitado, el fichero ".s" ya contiene código de las excepciones.



4. Presentar el simulador QT-SPIM

- Tras la ejecución aparece en la consola “Hola mundo”



5. Ejercicios

- Ejercicios propuestos:

- Descargar e instalar el simulador QT-SPIM.
- Descargar el código de la práctica y descomprimirlo en un directorio.
- Ejecutar y comprobar los resultados de los ejercicios de ejemplo incluidos en el directorio examples.
- Escribir un programa que devuelva el número de la posición 5 de la sucesión de Fibonacci.Compilarlo, ejecutarlo y comprobar que es correcto.