Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 – Arquitectura de Computadores (I/2020)

Proyecto Semestral: Entrega Práctica 03

Computador básico con input

Fecha de entrega: Viernes 29 de Mayo a las 20:00 horas

1. Objetivo

Para esta entrega tendrán que mejorar una vez más el computador básico desarrollado durante las entregas pasadas. Esto es agregando los componentes *Adder* y *SP*, y adicionalmente los selectores de la RAM, tanto de entrada como de dirección, el selector del PC y el selector de input.

En la Figura 1 se muestra el diagrama del computador básico modificado que tendrán que diseñar para esta entrega.

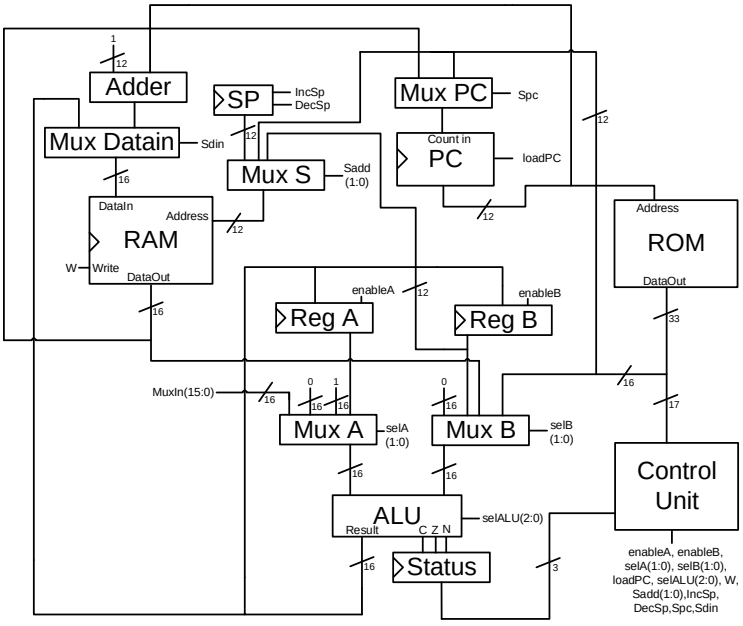


Figura 1: Computador básico.

Como se puede apreciar el computador consta de un registro *Stack Pointer*[1], un registro *Status*, un sumador *Adder*, los multiplexores *Datain*, *S*, *IN* y *PC*, además de un bus de entrada.

[1]Considere que si el contador de su implementación parte en cero, será necesario que al principio de cada programa se decremente en una unidad *SP* de manera que este parta en 111111111111.

Como se puede apreciar en la Figura 3, el contenido del bus de entrada esta determinado por el multiplexor *MuxIn*, que dependiendo de los 16 bits del literal selecionará una entrada distinta.

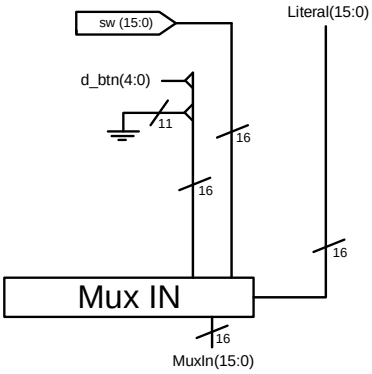


Figura 2: I/O detallado.

Para estandarizar el uso de los componentes, se reservarán los siguientes puertos para entradas:

Puerto	Input
0	Switches
1	Botones
*	Nada

Figura 3: Tabla de puertos Input.

2. Instrucciones

A continuación las nuevas instrucciones que deberá soportar su computador:

```
MOV A,(B)     (cargar Mem[B] en A)
    B,(B)     (cargar Mem[B] en B)
    (B),A     (guarda A en Mem[B])
    (B),Lit   (guarda Lit en Mem[B])

ADD A,(B)     (guarda A+Mem[B] en A)
    B,(B)     (guarda A+Mem[B] en B)

SUB A,(B)     (guarda A-Mem[B] en A)
    B,(B)     (guarda A-Mem[B] en B)

AND A,(B)     (guarda A and Mem[B] en A)
    B,(B)     (guarda A and Mem[B] en B)

OR  A,(B)     (guarda A or Mem[B] en A)
    B,(B)     (guarda A or Mem[B] en B)
```

```
XOR A,(B)      (guarda A xor Mem[B] en A)
    B,(B)      (guarda A xor Mem[B] en B)

NOT (B),A      (guarda not A en Mem[B])

SHL (B),A      (guarda shift left A en Mem[B])

SHR (B),A      (guarda shift right A en Mem[B])

INC (B)        (incrementa Mem[B] en una unidad)

CMP A,(B)      (hace A-Mem[B])

PUSH A          Mem[SP] = A, SP--
     B          Mem[SP] = B, SP--

POP  A          SP++, A = Mem[SP]
     B          SP++, B = Mem[SP]

CALL Dir        Mem[SP] = PC, PC = Dir, SP--

RET             SP++, PC = Mem[SP]

IN   A,Lit      A = Input[Lit]
     B,Lit      B = Input[Lit]
     (B),Lit    Mem[B] = Input[Lit]
```

Tablas de verdad componentes

A continuación se mostrará la tabla de verdad de algunos de los componentes del computador de esta entrega.

- `SP`

Entradas			Salidas
clock	up	down	dataout
Flanco Subida	*	*	dataout
Flanco Subida	1	0	dataout + 1
Flanco Subida	0	1	dataout - 1

Figura 4: Tabla de verdad de SP.

- `Adder`

Entradas		Salidas
a(11:0)	b(11:0)	result(15:0)
*	*	a + b

Figura 5: Tabla de verdad de Adder.

3. Desarrollo

Para desarrollar esta entrega se recomienda a los grupos realizar los siguientes pasos.

1. **Tener completada la entrega 2:** Partir de la base que se tiene todos los requerimientos de dicha entrega para poder avanzar correctamente en el desarrollo de esta entrega. En caso de que falte algún componente o instrucción no testeada, el grupo debe asegurarse de completar y testear el computador de la entrega pasada.

2. **Componentes:** Crear e importar los componentes nuevos del diagrama del computador en el archivo Basys3, procurando conectar correctamente las señales de entradas y salidas.

3. **Opcode y Señales:** Analizar y plantear si la forma en como la *ROM* entrega las instrucciones a la *Control Unit* es suficiente para agregar las nuevas instrucciones. Se recomienda ampliar las tablas que identifiquen el código de operación y sus respectivas señales asociadas.

4. **Comportamiento de los Componentes:** Completar los comportamientos específicos de cada uno de los componentes a desarrollar. Estos son: *Adder*, *Registro SP*, *MUX Datain*, *MUX S*, *MUX PC* y *MUX IN* .

   a) *Adder*: Un *Adder* que dados dos valores de 12 bits da el resultado de su suma como un valor de 16 bits.

   b) *Registro SP*: Este registro indica la última dirección utilizada en el stack. Tiene la funcionalidad de incrementar o decrementar en uno su valor.

   c) *MUX Datain*: selector que selecciona el dato de entrada de la *Main Memory,* ambos de 16 *bits*.

   d) *MUX S*: selecciona la dirección a que apuntará la *Main Memory*, los valores de selección de 12 *bits*.

   e) *MUX PC*: selector que seleccionen entre dos entradas para el valor de entrada del PC, cada una de 12 *bits*.

   f) *MUX IN*: selector que seleccionen entre tres inputs de 16 bits, sería el litetral, los *switches* y la señal de los botones (representada como *d_btn* en el dibujo).

5. **De Assembly a Binario:** Transformar los test a evaluar de lenguaje assembly a binario, generando un archivo ROM para cada uno de los test. Esto se podrá hacer manualmente o automáticamente, es decisión suya elegir como hacerlo.

6. **Simulación:** Finalmente, teniendo todo lo anterior listo, deberán simular y observar si los valores transitivos y finales de los display corresponden a los valores esperados o no. Para esto se le estregará un archivo de simulación listo con explicaciones de como deberán probar sus test.

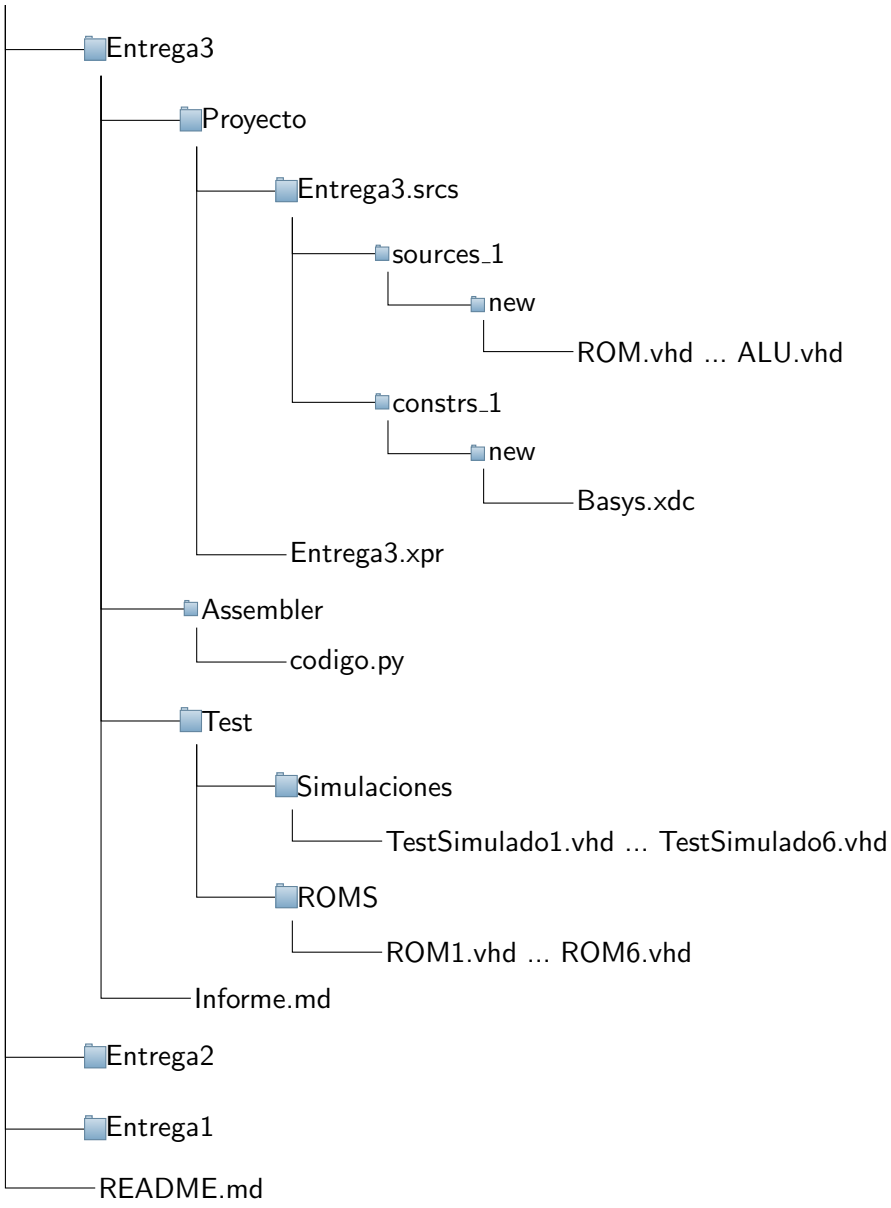
4. Requerimientos

A continuación se describen los requerimientos de esta entrega y el puntaje asociado a cada item:

- Para implementar declaraciones condicionales **solamente** se permite hacer uso de bloques `with/select`. El uso de los *statements process*, `case` e `if/else` quedan absolutamente prohibidos. Esto porque se privilegia el uso de selectores y operaciones lógicas básicas para el desarrollo del proyecto. **Para esta entrega, queda estrictamente prohibido utilizar cualquier tipo de librería aritmética que simplifique la tarea de la suma**. ***Nota:*** Esto solo aplica a los componentes a desarrollar, en los archivos de simulación se permite el uso de process.
- Incluir el archivo .xpr de su proyecto en su entrega (no incluirlo implicará un descuento por no cumplir con lo pedido).
- (6 pts) Se evaluara el contenido dentro de la entidad `Basys3.vhd`
  - (3 pts) Test correctos, bien visualizados y bien simulados en la Basys3:
    - (1 pts) Test 1
    - (0.5 pts) Test 2
    - (0.5 pts) Test 3
    - (0.5 pts) Test 4
    - (0.5 pts) Test 5
  - (0.5 pts) Correcta arquitectura de la CPU.
  - (0.5 pts) Uso correcto de todos los componentes de la CPU.
  - (0.5 pts) Correcta implementación de la *Control Unit* con las tablas de opcode y señales consistentes.
  - (0.5 pts) Correcta conexión de los comportamiento de la entidad Basys3 para desarrollar el problema.
  - (1 pts) Contenido del informe.
  - Puede crear más *sources* para facilitar el problema.
- Bonus
  - (0.5 pts) Test 6
  - Habiendo realizado el bonus ustedes obtendrán 0.5 ptos extras en esta entrega, pudiendo obtener como nota máxima de 7.5
- **Incluir su informe en formato Markdown :** En este debe un archivo llamado `Informe.md` que describa el trabajo realizado por cada uno de los miembros de su grupo. Se debe indicar específicamente que hizo cada integrante del grupo y deben explicar que fue lo mas difícil para el grupo en la entrega.

  Además, se deberá incluir resultados de los archivos de simulación, junto con los archivos ROM.vhd asociados, mostrando que su entidad efectivamente resuelve el problema. Para esto basta con mostrar el valor del display al final de la simulación que deberá mostrar los 8 bits menos significativos de cada registro (disA y disB serán el registro A, disC y disD serán el registroB. Los 4 bits menos significativos de cada registro son mostrados en disB y disD). **Nota:** se recomienda hacer más casos para probar su circuito, pero a la hora de escribir el informe solo basta con mostrar cada uno de los Test.
- **Formato del repositorio:** El formato del respositorio debe ser el siguiente:

- proyecto-entrega-grupo-X



NOTA: La carpeta Assembler es opcional, solo en caso de que hagan un programa para ello. Si traspasan los test a código de máquina manualmente no se debe subir, y lo deben especificar en su informe. **EL NO SEGUIR ESTE FORMATO DE REPOSITORIO PUEDE SIGNIFICAR UN DESCUENTO EN SU CALIFICACIÓN FINAL.**

5. Entrega

Deben entregar:

- La carpeta con su proyecto de Vivado. En el caso de la carpeta del proyecto, deben subir solo la carpeta `.srcs`, el archivo `.xpr` y el archivo `Basys3.xdc` dentro de la carpeta new de los constrains (ver el diagrama de más arriba).

- Una carpeta Test que contenga dos carpetas. Una carpeta de *Simulaciones* con sus archivos de simulaciones para cada test y una carpeta *ROMS* con sus archivos ROM para cada test
- Su archivo *informe.md*

La entrega del proyecto (código, informe y adicionales) es por medio del repositorio en Github tal que el Viernes 29 de Mayo a las 20:00 horas deben tener en la rama Master todos los archivos correspondientes a su grupo.

Antes de la entrega, se subirán los archivos para la evaluación en una issue en el Syllabus del curso. Toda acotación especial se debe incluir en el informe de cada entrega.

6. Evaluación de pares

La evaluación de pares estará activa durante la entrega y hasta el miércoles 27 de Mayo a las 23 horas. El link es el siguiente:

`https://forms.gle/rw1YrCuz778WbjbJA`.

Esto con el objetivo que nos informen si algún integrante de su grupo no se comunica o no se compromete con el proyecto para que, como equipo docente, poder intervenir antes de la entrega y encontrar soluciones posibles para el correcto desarrollo de la misma.

7. Ejemplos

Considere los siguientes ejemplos, los cuales corresponden a diversos programas que podría correr su computador:

- Programa 1:

```
DATA:

CODE:                   // Shift left rotate

MOV B,0                 // Puntero en 0
MOV A,8000h             // 1000000000000000b a A
MOV (B),A               // Guardar numero

shl_r:
 MOV A,0                // 0 a A
 OR A,(B)               // Recuperar numero
 SHL (B),A              // Guardar shift left de numero
                        // Si carry == 1
 JCR shl_r_carry        // Recuperar bit
 JMP shl_r_end          // No hacer nada
 shl_r_carry:
  INC (B)               // Agregar el bit perdido
 shl_r_end:
  JMP shl_r             // Repetir
```

- Programa 2:

```
DATA:

 arr    5
        Ah
        1
        3
        8
        5
 n      6
 r      0

CODE:                   // Sumar arreglo

MOV B,arr               // Puntero arr a B

siguiente:
 MOV A,(n)              // Restantes a A
 CMP A,0                // Si Restantes == 0
 JEQ end                // Terminar
 DEC A                  // Restantes --
 MOV (n),A              // Guardar Restantes
 MOV A,(r)              // Resultado a A
 ADD A,(B)              // Resultado + Arr[i] a A
 MOV (r),A              // Guardar Resultado
 INC B                  // Puntero en B ++
 JMP siguiente          // Siguiente

end:
 MOV A,(r)              // Resultado a A
 JMP end
```

- Programa 3:

```
DATA:

CODE:                   // Hack al stack

MOV A,2                 // 2 a A
PUSH A                  // Guarda A
MOV A,0                 // |
NOT B,A                 // | Puntero al primero en el stack a B
INC (B)                 // Primero en el stack++
POP A                   // Recupera A incrementado

end:
 JMP end
```

- Programa 4:

```
DATA:

CODE:                   // Swap con stack

MOV A,3                 // A = 3
MOV B,5                 // B = 5

PUSH A                  // |
PUSH B                  // |
POP A                   // |
POP B                   // | Swap con Stack
```

- Programa 5:

```
DATA:
CODE:                   // Subrutinas simples
MOV A,3                 // 3 a A
MOV B,2                 // 7 a B
CALL add                // A + B a B
MOV A,1                 // 1 A A
CALL add                // A + B a B
MOV A,7                 // 7 a A
CALL sub                // A - B a B
MOV A,B                 // B a A
fin:
JMP fin
add:
 ADD B,A                // A + B a B
RET
sub:
 SUB B,A                // A - B a B
RET
```

- Programa 6:

```
DATA:

CODE:                   // Subrutinas anidadas

MOV A,7
MOV B,1

CALL resta

```

```
fin:
 JMP fin

suma:
 XOR B,A                // Bits que no generan carry a B
 PUSH B                 // Guardar bits que no generan carries
 XOR B,A                // Recuperar segundo sumando
 AND A,B                // Bits que generan carry a A
 POP B                  // Recuperar bits que no generan carries
 CMP A,0                // Si carries == 0
 JEQ suma_fin           // Terminar
 SHL A                  // Convertir bits a carries en A
 CALL suma              // Sumar carries
 suma_fin:
 MOV A,B                // Resultado a A
RET

comp2:
 NOT A                  // Negado de A a A
 INC A                  // A++
RET

resta:
 PUSH A                 // Guarda minuendo
 MOV A,B                // Sustraendo a A
 CALL comp2             // Complemento a 2 del sustraendo a A
 MOV B,A                // Complemento a 2 del sustraendo a B
 POP A                  // Recupera minuendo
 CALL suma              // Suma de minuendo y complemento a 2 del sustraendo a A
RET
```

- Programa 7:

```
DATA:

CODE:                   // Sumar inputs

MOV B,0                 // Puntero en 0
IN (B),0                // Guardar switches
IN A,2                  // Nada a A
ADD A,(B)               // Sumar inputs
IN B,8000h              // Nada a B
ADD A,B                 // Sumar inputs

MOV B,0                 // Puntero en 0
IN (B),1                // Guardar botones
ADD A,(B)               // Sumar inputs
IN (B),FFFFh            // Guardar nada
```

```
ADD A,(B)               // Sumar inputs

end:
 JMP end
```

- Programa 8:

```
DATA:
CODE:                          // Sumar switches | Velocidad de clock a "full"
PUSH B                         // Guardar puntero
input:
 CALL std_io_btn_wait          // Esperar cambio en botones
 IN (B),0                      // Ingresar arr[i]
 MOV A,(B)                     // Recuperar switches
 INC B                         // Incrementar puntero
 CMP A,0                       // Si Switches != 0
 JNE input                     // Siguiente input
POP B                          // Recuperar puntero
MOV A,0                        // Resultado = 0
sumar:
 PUSH A                        // Guardar resultado
 MOV A,(B)                     // arr[i] a A
 CMP A,0                       // Si arr[i] == 0
 JEQ sumar_fin                 // Terminar
 POP A                         // Recuperar resultado
 ADD A,(B)                     // Resultado + arr[i]
 INC B                         // Puntero++
 JMP sumar                     // Siguiente
 sumar_fin:
 POP A                         // Recuperar Resultado
end:
 JMP end
std_io_btn_wait:               // * en A, * en B                        //
 PUSH B                        // Guarda B                              //
 IN A,1                        // Estado actual                         //
 std_io_btn_wait_press_lp:                                             //
  IN B,1                       // Nuevo estado                          //
  CMP A,B                      // Si ==                                 //
  JEQ std_io_btn_wait_press_lp // Continuar                            //
 XOR B,A                       // Bits cambiados                        //
 std_io_btn_wait_release_lp:                                           //
  IN A,1                       // Nuevo estado                          //
  AND A,B                      // Bits an cambiados                    //
  CMP A,0                      // SI != 0                               //
  JNE std_io_btn_wait_release_lp// Continuar                           //
 MOV A,B                       // Bits cambiados a A                    //
 POP B                         // Recupera B                            //
RET                            // Retorna Bit(s) en A
```

8. Assembly

Esta es la lista de instrucciones separadas por entrega.

Entrega 2

MOV	A,B	guarda B en A
	B,A	guarda A en B
	A,Lit	guarda un literal en A
	B,Lit	guarda un literal en B
	A,(Dir)	guarda Mem[Dir] en A
	B,(Dir)	guarda Mem[Dir] en B
	(Dir),A	guarda A en Mem[Dir]
	(Dir),B	guarda B en Mem[Dir]
ADD SUB AND OR XOR	A,B	guarda A op B en A
	B,A	guarda A op B en B
	A,Lit	guarda A op literal en A
	B,Lit	guarda A op literal en B
	A,(Dir)	guarda A op Mem[Dir] en A
	B,(Dir)	guarda A op Mem[Dir] en B
	(Dir)	guarda A op B en Mem[Dir]
NOT SHL SHR	A	guarda op A en A
	B,A	guarda op A en B
	(Dir),A	guarda op A en Mem[Dir]
INC	A	incrementa A en una unidad
	B	incrementa B en una unidad
	(Dir)	incrementa Mem[Dir] en una unidad
DEC	A	decrementa A en una unidad
CMP	A,B	hace A-B
	A,Lit	hace A-Lit
	A,(Dir)	hace A-Mem[Dir]
JMP	Ins	carga Ins en PC
JEQ	Ins	carga Ins en PC si en el status Z = 1
JNE	Ins	carga Ins en PC si en el status Z = 0
JGT	Ins	carga Ins en PC si en el status N = 0 y Z = 0
JGE	Ins	carga Ins en PC si en el status N = 0
JLT	Ins	carga Ins en PC si en el status N = 1
JLE	Ins	carga Ins en PC Ins si en el status N = 1 o Z = 1
JCR	Ins	carga Ins en PC Ins si en el status C = 1
NOP		no hace cambios

Entrega 3

MOV	A,(B)	guarda Mem[B] en A
	B,(B)	guarda Mem[B] en B
	(B),A	guarda A en Mem[B]
	(B),Lit	guarda Lit en Mem[B]
ADD SUB AND OR XOR	A,(B)	guarda A op Mem[B] en A
	B,(B)	guarda A op Mem[B] en B
NOT SHL SHR	(B),A	guarda op A en Mem[B]
INC	(B)	incrementa Mem[B] en una unidad
CMP	A,(B)	hace A-Mem[B]
PUSH	A	guarda A en Mem[SP] y decrementa SP
	B	guarda B en Mem[SP] y decrementa SP
POP	A	incrementa SP y luego guarda Mem[SP] en A
	B	incrementa SP y luego guarda Mem[SP] en B
CALL	Ins	guarda PC+1 en Mem[SP], carga Ins en PC y decrementa SP
RET		incrementa SP y luego carga Mem[SP] en PC
IN	A,Lit	guarda Input[Lit] en A
	B,Lit	guarda Input[Lit] en B
	(B),Lit	guarda Input[Lit] en Mem[B]

9. Contacto

Cualquier pregunta sobre el proyecto, ya sean de enunciado, contenido o sobre aspectos administrativos deben comunicarse con los ayudantes creando preguntas en el foro de canvas del curso o directamente con los ayudantes:

- Felipe Valenzuela: frvalenzuela@uc.cl
- Matías López: milopez8@uc.cl
- Cristóbal Herreros ceherreros@uc.cl
- Raúl Del Río Jara rjdelrio@uc.cl

10. Integridad académica

Los alumnos de la Escuela de Ingeniería de la Pontificia Universidad Católica de Chile deben mantener un comportamiento acorde a la Declaración de Principios de la Universidad. En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Docencia de la Escuela de Ingeniería.

Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un alumno para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno, sin apoyo en material de terceros. Por "trabajo" se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros. Si un alumno copia un trabajo, obtendrá nota final 1,1 en el curso y se solicitará a la Dirección de Docencia de la Escuela de Ingeniería que no le permita retirar el curso de la carga académica semestral. Por "copia" se entiende incluir en el trabajo presentado como propio partes hechas por otra persona.

Obviamente, está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la referencia correspondiente.

Lo anterior se entiende como complemento al Reglamento del Alumno de la Pontificia Universidad Católica de Chile. Por ello, es posible pedir a la Universidad la aplicación de sanciones adicionales especificadas en dicho reglamento.