



IIC2343 – Arquitectura de Computadores (I/2018)

Entrega Proyecto 03

Computador básico completo

Fecha de entrega: Lunes 11 de Junio a las 23:59 horas

## 1. Objetivo

Para esta entrega tendrán que mejorar una vez más el computador básico desarrollado durante las entregas pasadas, además de modificar su *assembler* de manera que soporte todas las instrucciones y especificaciones detalladas más adelante.

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

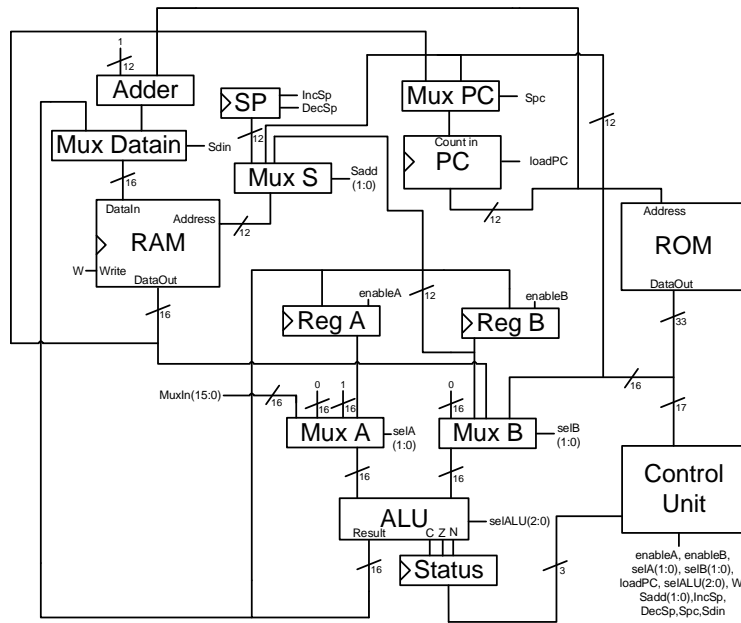


Figura 1: Computador básico.

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

<sup>1</sup>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 selecciona una entrada distinta.

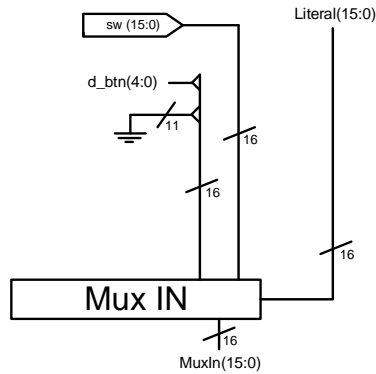


Figura 2: I/O detallado.

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 y *assembler*

Para más detalles sobre los cambios del assembler para esta entrega, revisar el enunciado *Assembler*. A continuación se muestran las instrucciones que debe soportar el 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]

### 3. Ejemplos

Considere los siguientes ejemplos, los cuales corresponden a diversos programas que su *assembler* debe ser capaz de compilar:

■ Programa 1:

```
1 DATA:
2
3 CODE:           // Shift left rotate
4
5 MOV B,0         // Puntero en 0
6 MOV A,8000h     // 1000000000000000b a A
7 MOV (B),A       // Guardar numero
8
9 shl_r:
10 MOV A,0        // 0 a A
11 OR A,(B)       // Recuperar numero
12 SHL (B),A      // Guardar shift left de numero
13               // Si carry == 1
14 JCR shl_r_carry // Recuperar bit
15 JMP shl_r_end  // No hacer nada
16 shl_r_carry:
17 INC (B)        // Agregar el bit perdido
18 shl_r_end:
19 JMP shl_r      // Repetir
```

■ Programa 2:

```
1 DATA:
2
3 arr      5
4          Ah
5          1
6          3
7          8
8          5
9 n        6
10 r       0
11
12 CODE:           // Sumar arreglo
13
14 MOV B,arr       // Puntero arr a B
15
16 siguiente:
17 MOV A,(n)       // Restantes a A
18 CMP A,0         // Si Restantes == 0
```

```

19 JEQ end           // Terminar
20 DEC A             // Restantes --
21 MOV (n),A         // Guardar Restantes
22 MOV A,(r)         // Resultado a A
23 ADD A,(B)         // Resultado + Arr[i] a A
24 MOV (r),A         // Guardar Resultado
25 INC B             // Puntero en B ++
26 JMP siguiente     // Siguiente
27
28 end:
29 MOV A,(r)         // Resultado a A
30 JMP end

```

■ Programa 3:

```

1 DATA:
2
3 CODE:             // Hack al stack
4
5 MOV A,2           // 2 a A
6 PUSH A           // Guarda A
7 MOV A,0           // |
8 NOT B,A           // | Puntero al primero en el stack a B
9 INC (B)           // Primero en el stack++
10 POP A            // Recupera A incrementado
11
12 end:
13 JMP end

```

■ Programa 4:

```

1 DATA:
2
3 CODE:             // Swap con stack
4
5 MOV A,3           // A = 3
6 MOV B,5           // B = 5
7
8 PUSH A           // |
9 PUSH B           // |
10 POP A           // |
11 POP B           // | Swap con Stack

```

■ Programa 5:

```
1 DATA:
2
3 CODE:           // Subrutinas simples
4
5 MOV A,3         // 3 a A
6 MOV B,2         // 7 a B
7 CALL add        // A + B a B
8 MOV A,1         // 1 A A
9 CALL add        // A + B a B
10 MOV A,7        // 7 a A
11 CALL sub        // A - B a B
12 MOV A,B        // B a A
13
14 fin:
15 JMP fin
16
17 add:
18  ADD B,A        // A + B a B
19  RET
20
21 sub:
22  SUB B,A        // A - B a B
23  RET
```

■ Programa 6:

```
1 DATA:
2
3 CODE:           // Subrutinas anidadas
4
5 MOV A,7
6 MOV B,1
7
8 CALL resta
9
10 fin:
11  JMP fin
12
13 suma:
14  XOR B,A        // Bits que no generan carry a B
15  PUSH B         // Guardar bits que no generan carries
16  XOR B,A        // Recuperar segundo sumando
17  AND A,B        // Bits que generan carry a A
18  POP B          // Recuperar bits que no generan carries
19  CMP A,0        // Si carries == 0
```

```

20 JEQ suma_fin          // Terminar
21 SHL A                 // Convertir bits a carries en A
22 CALL suma             // Sumar carries
23 suma_fin:
24 MOV A,B               // Resultado a A
25 RET
26
27 comp2:
28 NOT A                 // Negado de A a A
29 INC A                 // A++
30 RET
31
32 resta:
33 PUSH A                // Guarda minuendo
34 MOV A,B               // Sustraendo a A
35 CALL comp2            // Complemento a 2 del sustraendo a A
36 MOV B,A               // Complemento a 2 del sustraendo a B
37 POP A                 // Recupera minuendo
38 CALL suma             // Suma de minuendo y complemento a 2 del sustraendo a A
39 RET

```

■ Programa 7:

```

1 DATA:
2
3 CODE:                  // Sumar inputs
4
5 MOV B,0                // Puntero en 0
6 IN (B),0               // Guardar switches
7 IN A,2                 // Nada a A
8 ADD A,(B)              // Sumar inputs
9 IN B,8000h             // Nada a B
10 ADD A,B               // Sumar inputs
11
12 MOV B,0                // Puntero en 0
13 IN (B),1               // Guardar botones
14 ADD A,(B)              // Sumar inputs
15 IN (B),FFFFh           // Guardar nada
16 ADD A,(B)              // Sumar inputs
17
18 end:
19 JMP end

```

■ Programa 8:

```

1 DATA:
2
3 CODE:                // Sumar switches | Velocidad de clock a "full"
4
5 PUSH B                // Guardar puntero
6
7 input:
8   CALL std_io_btn_wait // Esperar cambio en botones
9   IN (B),0             // Ingresar arr[i]
10  MOV A,(B)             // Recuperar switches
11  INC B                 // Incrementar puntero
12  CMP A,0               // Si Switches != 0
13  JNE input             // Siguiente input
14
15 POP B                 // Recuperar puntero
16 MOV A,0               // Resultado = 0
17
18 sumar:
19   PUSH A               // Guardar resultado
20   MOV A,(B)            // arr[i] a A
21   CMP A,0              // Si arr[i] == 0
22   JEQ sumar_fin        // Terminar
23   POP A                // Recuperar resultado
24   ADD A,(B)            // Resultado + arr[i]
25   INC B                // Puntero++
26   JMP sumar            // Siguiente
27   sumar_fin:
28   POP A                // Recuperar Resultado
29
30 end:
31   JMP end
32
33 //////////////////////////////////Libreria std_io//////////////////////////////////////
34 //
35 std_io_btn_wait:       // * en A, * en B //
36   PUSH B               // Guarda B //
37   IN A,1               // Estado actual //
38   std_io_btn_wait_press_lp: // //
39   IN B,1               // Nuevo estado //
40   CMP A,B              // Si == //
41   JEQ std_io_btn_wait_press_lp // Continuar //
42   XOR B,A              // Bits cambiados //
43   std_io_btn_wait_release_lp: // //
44   IN A,1               // Nuevo estado //
45   AND A,B              // Bits an cambiados //
46   CMP A,0              // SI != 0 //
47   JNE std_io_btn_wait_release_lp // Continuar //

```



48	MOV A,B	// Bits cambiados a A	//
49	POP B	// Recupera B	//
50	RET	// Retorna Bit(s) en A	//
51			//
52	////////////////////////////////////		

## 4. Entrega

Deben entregar:

- El proyecto completo de la CPU, es decir, todos los archivos involucrados en su proyecto, sin las carpetas .caché, .hw, .runs o .sim (eso debe ser manejado a través del .gitignore).
- Una carpeta para el assembler.
- Un breve informe (incluyendo el número de grupo y el nombre de los integrantes) que contenga:
  - La especificación de la estructura de las instrucciones de su CPU (función de cada uno de los 33 bits de una instrucción).
  - Una tabla que muestre entradas y salidas de todos los multiplexores.
  - Una tabla anexa (puede ser un .csv) con todas las instrucciones soportadas por la CPU y su implementación de acuerdo a las señales de la Control Unit.
  - Una explicación de cómo ocupar su *assembler*, detallando paso a paso cómo usar su programa para ensamblar un archivo en *assembly* y generar el archivo de salida para insertar en la ROM.
  - Un párrafo por persona que especifica qué hizo cada integrante del grupo durante la entrega y una sección que explique qué fue lo más fácil y lo más difícil para el grupo en la entrega.

La entrega del proyecto (código, informe y adicionales) es por medio del repositorio en Github tal que el Lunes 11 de Junio a las 23:59 horas deben tener en la rama Master todos los archivos correspondientes a su grupo.

Entregas atrasadas serán **penalizadas con 0.5 puntos** por cada hora (o fracción) de atraso.

Para entrega 2 y 3: El día anterior a la entrega, se subirán los archivos para la evaluación tanto en la página del curso como en el Syllabus del curso. Son 6 archivos que corresponden a algoritmos en Assembly para evaluar. Deben generar los archivos .bit para cada uno de los archivos y subirlos a sus repositorios. Toda acotación especial se debe incluir en el informe de cada entrega o en un README.

## 5. Puntajes

A continuación se describe el puntaje asociado a cada ítem:

La entrega consistirá en mostrar los diferentes tests en el display. El display, al igual que en la entrega anterior, deberá mostrar los 8 bits menos significativos de cada registro. Entonces, se evaluará que la secuencia mostrada sea correcta. Si es correcta, se obtiene el puntaje completo, en caso contrario no se obtiene puntaje.

Cada ítem de la tabla está asociado a un ponderador. En la entrega cada ítem se puede evaluar entre 0 y 1 y ese valor se multiplica con el ponderador asociado. En el caso del descuento y bonus, se puede evaluar entre 0 y 0,5.

0.75	0.75	0.75	0.75	1	1	0.5	0.5
Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Formalidad	Informe

Figura 4: Tabla de puntajes.

## 6. Contacto

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

- Francesca Lucchini: flucchini@uc.cl
- Felipe Pezoa: fipezoa@uc.cl
- Hernán Valdivieso: hfvaldivieso@uc.cl
- Luis Leiva: lileiva@uc.cl

## 7. Evaluación

Cada entrega del proyecto se evaluará de forma grupal y se ponderará por un porcentaje de coevaluación para calcular la nota de cada alumno.

Dado lo anterior, dentro de las primeras **24 horas** posteriores a cada entrega, **todos los alumnos** deberán completar de forma **individual y obligatoria** el formulario web que los ayudantes pondrán a su disposición, repartiendo un máximo de 4 puntos (aunque cambia según la cantidad de estudiantes en el grupo; el máximo corresponde al número de compañeros). Los puntajes pueden tener hasta un decimal y una diferencia máxima entre el mayor y menor puntaje de 1. La suma de todos los puntos obtenidos por el integrante,  $sp$ , será utilizada para el cálculo de la nota de cada entrega, lo que puede hacer que este repruebe el curso.

La nota de cada entrega se calcula de la siguiente forma (con un máximo de 7,5):

$$NotaEntrega_{individual} = \min(k_g \times NotaEntrega_{grupal}, NotaEntrega_{grupal} + 0,5)$$

donde,

$$k_g = \frac{sp+3}{7}$$

y  $sp$  puede variar de acuerdo a la cantidad de estudiantes que conforman el grupo.

Los alumnos que no cumplan con enviar la coevaluación en el plazo asignado tendrán un **descuento individual de 0.5 puntos** en su nota de la entrega correspondiente.

## 8. 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.