Nro. ord.	Apellido y nombre	L.U.	#hojas

Organización del Computador I - **Parcial**

2^{do} Cuatrimestre 2017

Ej.1	Ej.2	Ej.3	Nota

Corrector:

Aclaraciones

- Anote apellido, nombre, LU y numere *todas* las hojas entregadas, entregando los distintos ejercicios en hojas separadas.
- Cada ejercicio será calificado con una de las siguientes tres notas: Bien, Regular o Mal. La división de los ejercicios en incisos es meramente orientativa. Los ejercicios se calificarán globalmente.
- El parcial **es a libro abierto**. Pueden tener cualquier material en papel, así también como una calculadora que no cuente con conexión a internet.
- **Importante:** Justifique sus respuestas. Las soluciones a ejercicios de la práctica que se utilicen deben ser incluídas en el examen.
- Un resultado sin justificar equivale a un ejercicio no resuelto.
- El parcial se aprueba con al menos dos ejercicios Bien y uno Regular. Para obtener un Regular es necesario demostrar conocimientos sobre el tema del ejercicio. Además, el ejercicio 1 debe estar al menos Regular.

Ejercicio 1 Por el Orga 1 Friday están en oferta las compuertas **or** y **not** a 100 % de descuento. Esto quiere decir que son gratis. El descuento tenía que ser del 10 % pero hubo un error en la máquina que usaron para cargar el descuento y por lo tanto nos ofrecen esas compuertas. Compramos entonces muchísimas de cada una, y necesitamos, sólo usando esas compuertas, armar una ALU de 5 bits.

a) Implemente las compuertas **and**, **xor**, **nand** que tienen dos entradas y una salida.(Bit a bit)

b) Implemente **xor** bit a bit para números de 5 bits.

c) Implemente **and** y **xor** lógico para números de 5 bits. Para esto, cualquier input distinto de cero se considera true y el cero se considera false, el resultado debe ser 00000 si es false y 00001 si es true.

d) Implemente la suma binaria en complemento a 2 de 5 bits con flags de overflow, carry, negativo y cero. Se dispone de un sumador sin flags de 5 bits que pueden utilizar sin implementar.

e) Realice una ALU con las funcionalidades anteriores y flags de carry, overflow, negativo y cero. Para la suma, los flags deben ser los usuales de suma en complemento a 2. Para el Xor, tanto el carry como el overflow deben quedar en 0, y el negativo y el cero tener el comportamiento esperado. El valor de los flags luego de las operaciones **and** y **xor** lógicos serán ignorados por lo que pueden tomar cualquier valor.

Para conectar la salida de un circuito de 5 bits con la entrada de otro circuito de 5 bits, pueden usar una sóla línea con un número 5 escrito al lado de dicha línea.

Para la resolución de este ejercicio pueden armar circuitos utilizando las compuertas disponibles, o reutilizar circuitos previamente armados durante la resolucion del mismo, y utilizarlos a modo de caja negra.

Aclaración: El Xor bit a bit y el Xor lógico son operaciones distintas.

Ejercicio 2 Tenemos una computadora con una memoria de 512MB y palabras de 32 bits, con direccionamiento a palabra, 16 registros de 32 bits, y las siguientes instrucciones:

- `movmr p, q` (siendo p una dirección de memoria y q un registro)
- `movrm p, q` (siendo p un registro y q una dirección de memoria)
- `movr p, q` (siendo p y q registros)
- `add p, q` (siendo p y q registros)
- `sub p, q` (siendo p y q registros)
- `push p` (siendo p una dirección de memoria, se *"pushea"* el contenido a una pila disponible en la memoria)

- `pushr p` (siendo p un registro, se *"pushea"* el valor del registro a la pila)
- `popr p` (siendo p un registro, se *"popea"* el tope de la pila al registro)

a) Si tenemos instrucciones de 32 bits, podemos codificar este set de instrucciones?

b) En caso afirmativo, dar un set de instrucciones, caso negativo, decidir si existe algún subconjunto de instrucciones que se pueden eliminar para no perder funcionalidad (poder hacer las mismas cosas, reemplazando las instrucciones eliminadas por secuencias de instrucciones no eliminadas) y que se pueda codificar en 32 bits.

   Cada instrucción de la lista podrá ser eliminada, pero no modificada. Por ejemplo, de eliminar la instrucción pushr, no se podrá utilizar. Lo que **no se puede** es reducir la instrucción a poder pushear sólo de un único registro posible.

c) Suponiendo que un set de instrucciones para la arquitectura original (más allá de las modificaciones que se puedan haber hecho en el punto B) se debe implementar para memoria de 64 MB en lugar de 512 MB, cual es la mínima cantidad de bits que se necesitan para codificar todas las instrucciones? Dar el set de instrucciones con esta cantidad mínima de bits.

Ejercicio 3 Nos fue tan bien en el Orga 1 Friday que queremos ver qué tan buenos fueron los descuentos. Para eso, queremos calcular cuál fue el mayor descuento que obtuvimos en aquellos items que fueron una compra exitosa.

Esto es porque si bien algunas ofertas fueron muy buenas, lamentablemente otras no tanto, pese a tener grandes descuentos. Por ejemplo, las compuertas del ejercicio 1 salieron un poco defectuosas (probablemente de ahí que hayan sido gratis), por lo que preferimos no utilizar la arquitectura diseñada con esas compuertas, y trabajar con la arquitectura de la máquina **Orga 1**.

Podemos asumir que en cada posición de memoria contamos con un valor a partir del label *datos_ventas* y que se termina con un cero y que el `PC` comienza en *start*.

```
datos_ventas:   dw 0x0001
                dw 0x0321
                ...
                dw 0x0032
                dw 0x0000
start:
```

Estos valores, representan el monto (en términos absolutos, y no porcentuales) del descuento obtenido en cada uno de los items a considerar.

El objetivo, es escribir a partir del label *start*, el código mediante el cuál calculamos el monto del mayor descuento obtenido, para guardarlo en el registro R0.

El valor de la etiqueta *datos_ventas* se encuentra almacenado en el registro R0 al comenzar la ejecución del programa.

Los valores (0x0001, 0x0321, 0x0032) son sólo a modo de ejemplo.

Ejercicio 4 Nos pidieron realizar una computadora muy específica, llamada **BugBarato**, o **BB**, y de muy bajo costo. Es por esto que contamos con especificaciones muy acotadas. La teníamos terminada, pero resulta que el sitio donde hicimos el diseño cambió de dueños y ya no anda más. Quedaron únicamente las siguientes especificaciones.
Una computadora cuenta con 8 registros de uso general, direcciones de 5 bits, direccionamiento a palabra y tamaño de palabra de 16 bits. Su conjunto de instrucciones es el que sigue:

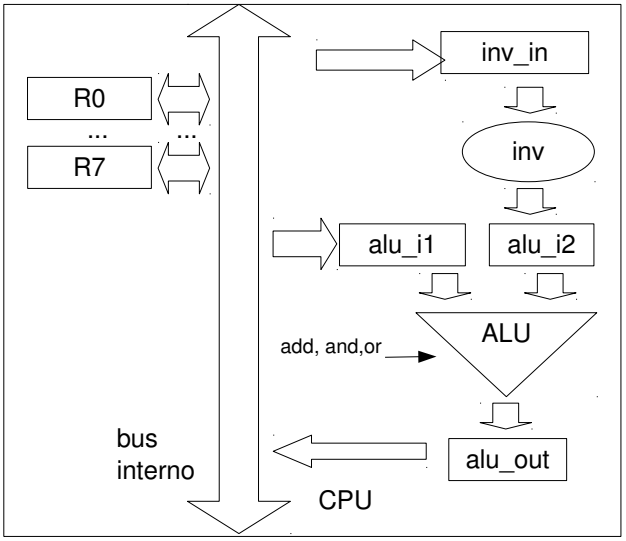
Instrucciones	Operandos	Efecto
add and/or	reg1,reg2 reg1,reg2	reg1 := reg1 + reg2 reg1 := reg1 and/or reg2
load/store be/bl	addr addr	carga/guarda el valor del registro R0 desde/hacia memoria PC := addr si R0=0/R0<0

El formato de la instrucción es de largo fijo y es:

add, and, or		
codOp	reg1	reg2
3 bits	3 bits	2 bits

be, bl, load, store	
codOp	addr
3 bits	5 bits

Sea la siguiente microarquitectura parcial del CPU:



Donde **inv** realiza calcula el inverso aditivo en formato complemento a dos.

a) Completar la microarquitectura de la computadora para soportar la implementación del conjunto de instrucciones. Agregar la cantidad de bits de cada bus.

b) Al márgen de $R0$, son todos los registros intercambiables en todas las instrucciones? Justificar adecuadamente.

c) Podríamos agregar la operación *sub* sin modificar el ancho ni estructura de operaciones ? Podríamos agregar alguna más ? Justificar adecuadamente.

d) Describir la secuencia de microinstrucciones que utiliza la unidad de control para:

   a. Ejecutar la instrucción

      `sub R0,R1`

      Que ejecuta $R0 := R0 - R1$

   b. Realizar el fetch de una instrucción