Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática
CI-0114 Fundamentos de Arquitectura
Profesor: Sergio Guillén
Grupo 006
I Ciclo 2022
Tarea 1

Esta tarea debe realizarse de forma individual.

Debe entregarse digitalmente mediante mediación virtual a más tardar el viernes 10 de Junio a las 11:59 pm. Cada hora de entregar tardía restara 1 punto a la calificación final de la tarea.

Esta tarea deberá realizarse en el lenguaje de descripción de hardware Verilog. La implementación deberá hacerse a nivel RTL (Register Transfer Level).

La tarea consiste en implementar algunos de los elementos básicos con que componen un sistema mínimo de computadoras (CPU + Memoria). Se implementarán los siguientes componentes:

1. Una unidad lógica aritmética (ALU)
2. Un archivo de registros (Register File)
3. Un módulo CPU principal que incluya la funcionalidad básica de búsqueda (fetch) y decodificación de una instrucción. Esta CPU deberá estar conectada a su respectiva memoria.

La memoria se direccionará con una granularidad de palabras (words) de 32 bits y tendrá una profundidad de 64Ki words (por lo cual, las direcciones serán de 16 bits). Los registros y las instrucciones serán de 32 bits (coincidiendo con el ancho de los datos de la memoria).

En el archivo de registro, se almacenan 8 registros de propósito general (R0-R7) de 32 bits cada uno. El modulo de archivo de registro tendrá 2 puertos (data + dirección) de salida (se pueden “leer” 2 registros al mismo tiempo) y 1 puerto de entrada (se puede escribir 1 único registro a la vez)

La ALU tendrá 2 operandos (señales) de entrada de 32 bits (aunque algunas operaciones podrían de requerir solo 1 de ellos) y el código de operación (opcode) a ser ejecutado. Tendrá también 2 señales de salida, una de 32 bits con el resultado de la operación y otra de 4 bits indicando el estado de la operación.

Estos 4 bits de estado de operación se asignan siempre que la ALU ejecute alguna operación lógica o aritmética (que corresponde las banderas del registro de estado o Program Status Word [PSW]) y son:

- C: Acarreo
- S: Signo
- O: Desbordamiento (overflow)
- Z: Cero

El módulo de CPU deberá detener la ejecución (transicionando a un estado de “Suspendido” o “Halted”) cuando la instrucción HLT sea identificada en la etapa de decodificación. La implementación de este CPU será **sin** pipeline, pero se utilizaran las 5 etapas básicas del pipeline (FE, DE, EX, MA, WB) para la ejecución de las instrucciones. Para esta tarea, la instrucción deberá ser únicamente decodificada (la etapa de decodificación descompondrá el Registro de Instrucción [IR] en los operandos que correspondan) pero no ejecutada. Las instrucciones de prueba deben de ser agregados directamente en la inicialización de la memoria en el archivo **memoria.v**.

Recomendaciones:

1. Utilice los flancos positivos del reloj para la máquina de estados de la unidad de control del CPU y flancos negativos para guardar los valores en el archivo de registro.
2. Incluya una señal write_en (habilitar escritura) en el archivo de registros.
3. Utilice un registro buffer de memoria para lecturas (MBR_R) y un registro independiente para escrituras (MBR_W). De esta manera no tendrá que preocuparse por posibles contenciones ni de lógica de 3 estados.
4. Utilice el modulo de memoria Mem_D32b_A16b (en memoria.v) como referencia para implementar su archivo de registros.
5. Implemente todas las operaciones de la ALU a nivel de comportamiento (use como referencia los opcodes provistos en alu.v)
6. Utilice 2 estados (_0 y _1, o bien cualquier otra nomenclatura) para cada una de las etapas de una instrucción.

A continuación, se detallan los Opcodes requeridos en esta etapa (solo deben decodificarse, aun no deben ejecutarse). Los opcodes de ALU deberán inyectarse directamente en la testbech de la ALU para verificar su funcionamiento

Forma General

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode					Dir			Operando A								Operando B																

OPD_NOP

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode					Dir (IMP)																										
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

OPD_HLT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode					Dir (IMP)																										
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(ALU, Lógica)
OPD_NOT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode					Dir (REG)								Dst																		
1	0	0	0	0	0	1	0	0	0	0	0	0	D	D	D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

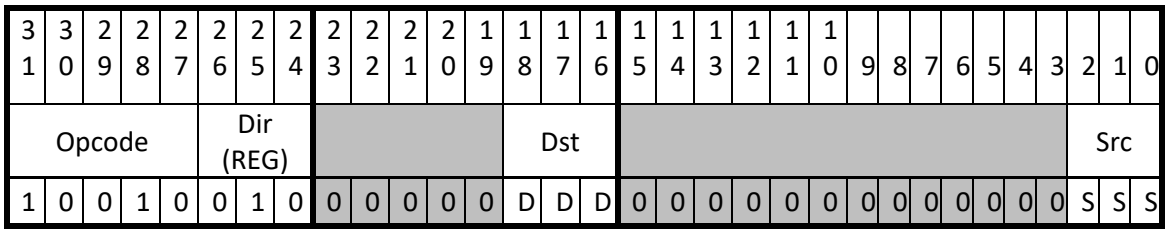
[Reg Dst] = ~[Reg Dst]

(ALU, Lógica)
OPD_AND

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode					Dir (REG)								Dst																Src		
1	0	0	0	1	0	1	0	0	0	0	0	0	D	D	D	0	0	0	0	0	0	0	0	0	0	0	0	0	S	S	S

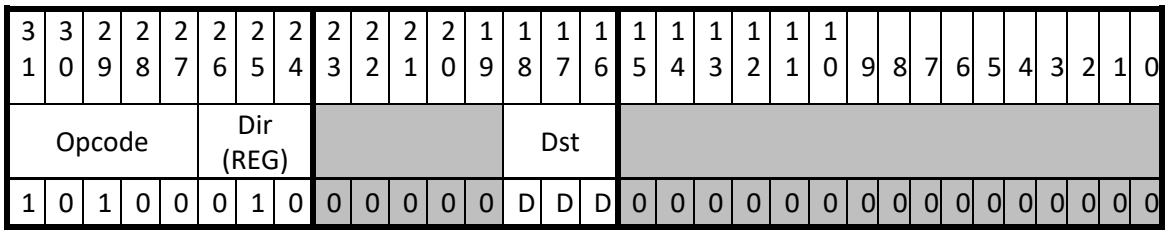
[Reg Dst] = [Reg Dst] & [Reg Src]

(ALU, Lógica)
OPD_OR



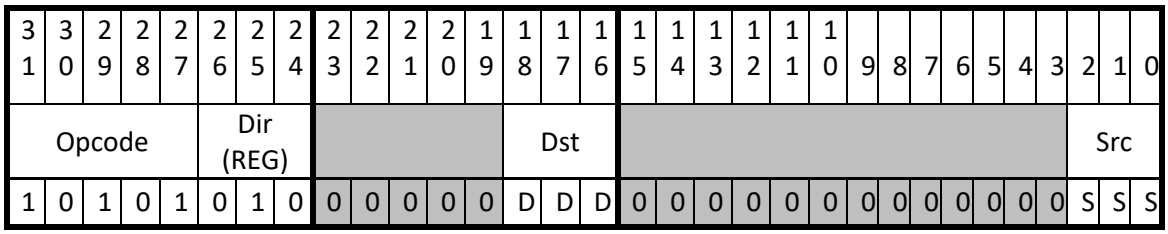
[Reg Dst] = [Reg Dst] | [Reg Src]

(ALU, Aritmética)
OPD_NEG



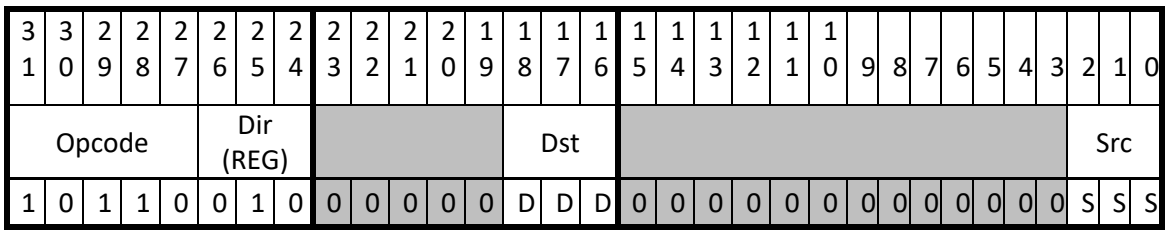
[Reg Dst] = -[Reg Dst]

(ALU, Aritmética)
OPD_ADD



[Reg Dst] = [Reg Dst] + [Reg Src]

(ALU, Aritmética)
OPD_SUB



[Reg Dst] = [Reg Dst] - [Reg Src]

Entregables:

1. Implementación de la ALU, en el archivo **alu.v** y su correspondiente testbech **alu_tb.v**. Debe actualizar el archivo provisto para cumplir con los requerimientos descritos en los puntos anteriores
2. Implementación de archivo de registros (**registros.v**) y su correspondiente testbench (**registros_tb.v**) que cumpla con los requerimientos descritos en los puntos anteriores
3. Implementación del módulo principal de CPU, en el archivo **cpu.v** y su correspondiente testbech **cpu_tb.v**. Debe actualizar el archivo para cumplir con los requerimientos descritos en los puntos anteriores

Calificación:

1. Implementación de alu.v [--15%--]
2. Implementación de registros.v [--15%--]
3. Implementación de cpu.v [--9%--]
4. 3 archivos testbech (.v), 1 para cada uno de los puntos anteriores [--21%--]
5. Los archivos .v compilan [--20%--]
6. El ejecutable generado completa su ejecución correctamente [--20%--]