PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2343 — Arquitectura de Computadores — 1/2018

I1

Viernes 6 de Abril de 2018

La interrogación se compone de 3 grupos de preguntas. En cada grupo de preguntas usted debe responder una pregunta. Si responde más de una, será criterio exclusivo del corrector qué nota será la considerada y esto no tendrá apelación posible.

Grupo I.

1. El computador básico solo trabaja con números enteros, sin embargo, para muchas aplicaciones es útil poder trabajar con números decimales. En consecuencia, usted deberá añadir soporte para números decimales al computador básico.

   1.1) Indique un esquema de números decimales que podría emplear en el computador básico, mencionando sus ventajas y desventajas. **[2p]**
   Solución:
   Un esquema posible es el uso de precisión fija decimal. De este modo se utiliza un número fijo de bits para parte entera y parte decimal, tiene la ventaja de ser relativamente simple y barato de implementar pero la desventaja de estar limitado en el rango de números que puede abarcar. Otros esquemas pueden ser esquemas de punto flotante como IEEE754.

   1.2) Añada los componentes necesarios para trabajar con el esquema de números decimales argumentados en la parte **1.1**, indicando qué hace cada componente y cómo se conecta a las partes existentes del computador básico, detallando su interacción con este. Si bien no es necesario hacer a nivel de compuertas todos los componentes, no se aceptarán componentes mágicos como "Unidad de cómputo decimal". Sí se aceptará la `ALU`, registro de n-bits y otros componentes desarrollados en clases. **[4p]**
   Solución:
   Es posible añadir 4 registros de 8 bits, igual a los A y B del computador básico, que pueden ser llamados DAi, DAd, DBi y DBd, por Decimal (A o B) (integer/decimal). Dichos registros tendrán sus data in conectados a la salida de la ALU y sus salidas conectadas a una ALU de partes decimales y otra de partes enteras. Además, el carry-out de la ALU de partes decimales estará conectada al Carry-IN de la ALU de partes enteras. Las salidas de dichas ALUs realimentarán las rutas que se alimentaban originalmente de la salida de la ALU normal. Este esquema hace que se requieren 4 cargas para cargar los operandos decimales y 2 ciclos para guardar sus resultados. Además, modifica las operaciones enteras para que pasen sin operar en las ALUs de decimales. Otro approach es usar los registros y ALUs al mismo nivel de la ALU normal, incluso pudiendo reutilizar esta con los nuevos registros, y usar un mux de salida para seleccionar qué parte del resultado es la que se carga a registro o se guarda en memoria.

2. Responda las siguientes preguntas breves.

   2.1) ¿Es posible explicar la llamada y salida de subrutinas en función de otras instrucciones del computador básico?, es decir, explicar `CALL` y `RET` en función de otras instrucciones. Justifique. **[1p]**
   Solución:
   Si, es posible explicarlas en función de Jumps y POPs

   2.2) Indique qué problemas puede traer que el *stack* del computador resida en memoria principal y cómo podría solucionar dicho problema. Incluya un diagrama en caso de ser necesario. **[1p]**
   Solución:

El problema está en que el stack podría crecer tanto que pise memoria en uso por el programa. Una posible solución es agregar una memoria exclusiva para el stack. Idealmente se debe añadir un esquema de una memoria de stack que esté lado a lado con la RAM principal, su entrada address conectada a la salida de SP, su salida muxeada con la salida de la RAM y la misma entrada de datos de la RAM.

2.3) ¿Qué ventajas tiene IEEE754 sobre otras representaciones de números racionales? **[1p]**
Solución:
La precisión extendida, el aprovechamiento que la parte entera en decimal siempre va a ser 1 cuando se usa notación científica. Acuerdo global del formato, entre otras.

2.4) ¿Qué relación existe entre las bases binaria, octal y hexadecimal? ¿Hay alguna forma de alternar rápidamente entre estas bases? **[1p]**
Solución:
Octal y Hexadecimal son bases potencias de binaria. La forma de pasar entre ellas es tener el número en binario y agrupar de a 3 bits para octal o 4 para hexa. De estas a binario cada símbolo se convierte en 3 o 4 bits respectivamente.

2.5) Interprete los números 1089 y 0xFBF6 en IEEE754 y realice la multiplicación **[1p]**
Solución:
$Bin(1089) = Signo : 0Exp : 00000000Significante : 00000000000010001000001$
$Hex(0xFBF6) = Signo : 0Exp : 00000000Significante : 00000001111101111110110$
Notamos que los exponentes son los mismos, y el resultado será positivo.
Multiplicamos los significantes, el resultado de dicha multiplicación es $10000101111110100010111011 0$.
La primera parte se mantiene, los últimos 4 bits se pierden por falta de precisión.
Siguiendo las reglas de la multiplicación en *float*, tenemos que el nuevo exponente será $10000001 = 0 + 0 - 127$
Finalmente el resultado será $01000000110000101111110100010111$

2.6) ¿Sería posible construir una `RAM` que permita escribir en una dirección y leer otra? Justifique. **[1p]**
Solución:
Si, ya que, si la suponen hecha de registros, siempre la salida es determinada por un MUX y el write-enable pasaría por un Decoder de forma normal, sólo se agregaría otro bus de direcciones.

Grupo II.

1. Un computador posee 2 registros de 1 bit (A y B) y 2 instrucciones `MOV` (que puede ser entre `REG1` y `REG2`, `REG` y `LIT`, `DIR` y `REG`; y `REG` y `DIR`) y `NAND A,B`, que hace `NOT(AND(A, B))` y almacena el resultado en `A`. Basándose en esto, genere un código que realice:

   1.1) `A = NOT A` **[1.5p]**

   Solución:

```
MOV B, A
NAND A,B
```

   1.2) `DIR = A OR B` **[1.5p]**

   Solución:

```
MOV B,A
NAND A,B
MOV (0),A
MOV A,B
NAND A,B
MOV B,(0)
NAND A,B
MOV (0),A
```

   1.3) `A = A + DIR` (sin carry) **[1.5p]**

   Solución:

```
MOV (2),A
MOV B,(0)
NAND A,B
MOV (1),A
MOV B,(0)
MOV A,(1)
NAND A,B
MOV (3),A
MOV A,(1)
MOV B,(2)
NAND A,B
```

   1.4) `B = A - B` (sin carry) **[1.5p]**

   Solución:

```
MOV (2),A
MOV (0),B
MOV B,(0)
NAND A,B
MOV (1),A
MOV B,(0)
MOV A,(1)
NAND A,B
MOV (3),A
MOV A,(1)
MOV B,(2)
NAND A,B
MOV B,A
```

2. Responda las siguientes preguntas:

2.1) Construya un circuito que obtenga la siguiente tabla de verdad, pero solo usando compuertas `OR` y `NOT`: **[3p]**

R	**S**	**Q(t+1)**	**¬Q(t+1)**
0	0	-	-
1	0	0	1
0	1	1	0
1	1	Q(t)	¬Q(t)

Cuadro 1: Tabla de verdad del Latch RS.

Solución: Un circuito que cumple lo pedido es el siguiente.

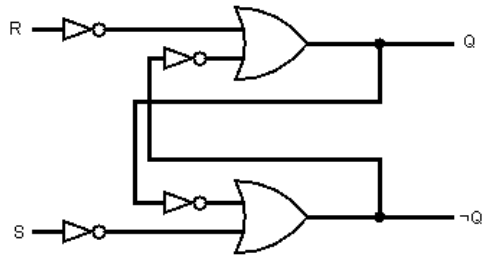

Figura 1: textitLatch RS con compuertas `OR` y `NOT`.

Este se deduce a partir de la **Ley de Morgan**. En el *latch* RS tradicional, las componentes siguen las siguientes fórmulas de lógica proposicional:

- `NOT(R AND ¬Q)`
- `NOT(S AND Q)`

Al desarrollar ambos términos con la Ley de Morgan, se obtiene lo siguiente:

- `¬R OR Q`
- `¬S OR ¬Q`

Que es lo que finalmente representa el circuito anterior.

2.2) Diseñe un registro de 1 byte haciendo uso de *flip-flops* D y una señal de control `L` que habilite la sobreescritura del estado Q solo cuando `L = 1`. Luego, explique cómo puede modificar el circuito realizado para acceder individualmente (tanto para lectura como escritura) a cada uno de los bits del componente. **[3p]**

Solución: En primer lugar, se muestra el diagrama para un solo bit del registro, en particular, el bit menos significativo.

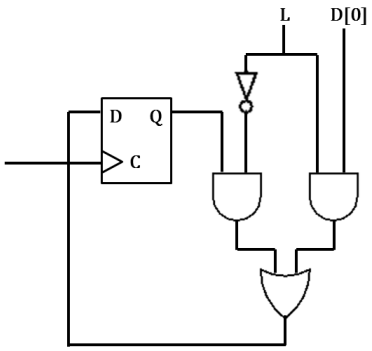

Figura 2: Diagrama del almacenamiento del bit menos significativo del registro.

En este caso, `D[0]` representa el bit menos significativo que se busca almacenar, mientras que `L` es la señal de carga. Así, se obtiene la siguiente tabla de verdad.

L	D[0]	Q
0	0	Q
0	1	Q
1	0	0
1	1	1

Cuadro 2: Tabla de verdad del almacenamiento del bit menos significativo del registro.

De esta forma, se evidencia el cumplimiento del primer objetivo: mantener el estado de un bit para `L = 0`. Para formar el byte, consiguientemente, basta por realizar este circuito para todos los bits del registro, conectados con todos los bits de `D`.

Para extender la lectura y la escritura a bits individuales del registro, existen muchas posibilidades, aquí se muestra una. En primer lugar, definimos un bus de control llamado S_{bit} que permita seleccionar el bit al que se desea acceder. Aquí, se debe cumplir que:

$$|S_{bit}| = \lceil \log_2(\text{Número de bits en el registro}) \rceil$$

Esto permitirá poder seleccionar todos los bits del registro. En este caso, se cumple que $|S_{bit}| = 3$.

Ahora, queremos que la señal `L` llegue **solo** al bit de interés. Para este propósito, podemos usar un Demux con entrada `L` y con bus de control igual a S_{bit}.

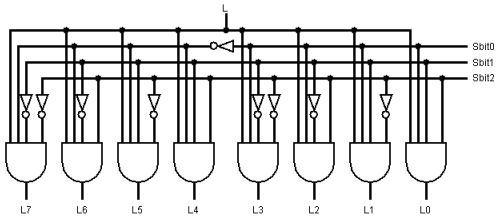

Figura 3: Demux con bus de control de 3 bits y señal de entrada `L`.

De esta forma, solo la señal $\texttt{L}_i$ va a ser igual a 1, habilitando la escritura del bit i del registro (sin importar el valor del resto de los bits que se encuentran en el bus de entrada).

Por último, necesitamos que el bus de salida solo posea el bit de interés. Para este propósito, usamos un Mux que tenga como entradas todos los bits del registro y que haga uso del mismo bus de control S_{bit}.

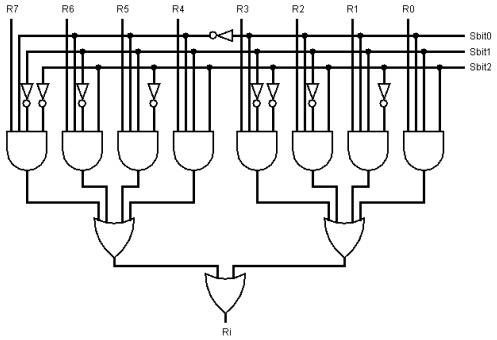

Figura 4: Mux con bus de control de 3 bits y bus de entrada `R` (valor del registro).

Finalmente, conectamos el bit resultante con un bus de 7 bits iguales a cero, de forma que el bus de salida del registro posea 8 bits (para respetar el formato del registro), siendo el bit menos significativo igual al seleccionado en un principio.

Nota: Para la evaluación no es relevante que hagan los diagramas del Mux y Demux, pero sí que expliquen su funcionalidad dentro de las modificaciones que realicen sobre el registro.

Grupo III.

1. Modifique el computador básico para que acepte el comando `MOV A,[DIR]`, que toma el valor almacenado en la dirección `DIR` y luego considera ese valor como una dirección, va a esa dirección y almacena su valor en `A`.
   Solución: Una posible solución consiste en agregar un registro interno temporal para este tipo de indirecciones, llamémoslo T de temporal. Este registro tendrá su entrada de datos del mismo bus de datos de A y B y su salida será conectada al MuxA del computador. Está de más agregar que la instrucción que se busca dar soporte no puede ser implementada en un ciclo de reloj debido al funcionamiento de la RAM.
   Lo que se hace es tomar el literal y cargar desde la RAM el valor almacenado en la dirección DIR y almacenarlo en el registro temporal T, luego en el siguiente ciclo se usa la salida de dicho registro como entrada para MuxAddress y se obtiene el valor solicitado.

2. Modifique el hardware del computador básico para que las instrucciones `RET` y `POP` tomen un solo ciclo.

   Solución: Una opción es hacer uso de un sumador de entradas `SP` y 1. Este puede tener su salida conectada a un registro llamado '`SP+1`' conectado al *clock* del computador básico, de forma que esté sincronizado con el resto de los registros. Luego, su salida puede estar conectada al `Mux Address` y, en caso de las operaciones `RET`

y `POP`, se configura *Sadd* para su selección. Luego, se puede obtener `Mem[SP+1]` de forma inmediata para su posterior escritura en el registro que corresponda (ya sea `PC`, `A` o `B`). El siguiente diagrama muestra esta idea:

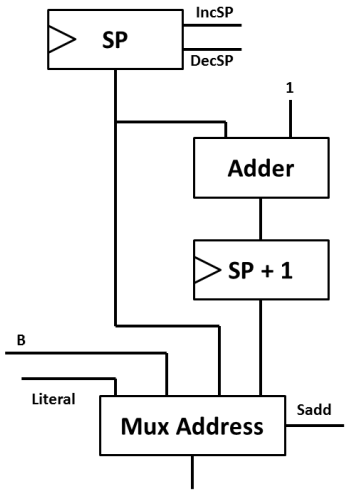

Figura 5: Diagrama del registro `SP+1`.

Esto, finalmente, permite reducir las operaciones `POP` y `RET` a un solo *opcode*, configurando correspondientemente al `Mux Address` para la selección del registro creado.