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

IIC2343 – Arquitectura de Computadores

Guía 1 – ISA – Solución propuesta

Profesor: Yadran Francisco Eterovic Solano

Ayudantes: Germán Leandro Contreras Sagredo (glcontreras@uc.cl)

Jurgen Dieter Heysen Palacios (jdheysen@uc.cl)

Nota al lector

El título dice 'solución propuesta' por una razón bien sencilla: Estos ejercicios pueden tener más de un desarrollo correcto. Lo que se pretende hacer aquí es mostrar un camino a la solución, sin excluir la posibilidad de rutas alternativas igual de correctas.

Preguntas

1. a. **(I2 - II/2015)** Compare las arquitecturas Harvard y Von Neumann desde el punto de vista del tiempo de ejecución de las instrucciones. Fundamente y explique claramente las diferencias.

   En la arquitectura de Harvard (utilizada para el computador básico estudiado en el curso) se tiene un tiempo promedio menor en ejecución si se compara con la arquitectura Von Neumann. Esto, debido a que en la primera (salvo por las instrucciones `POP` y `RET`) utiliza un ciclo por instrucción, mientras que Von Neumann utiliza al menos dos o tres (un ciclo donde se obtiene el *opcode* de la instrucción y se transfiere, otro donde se obtiene el literal y se transfiere -estas dos se podrían realizar en uno- y el último para ejecutar la instrucción en sí). Si bien hay instrucciones que se podrían seguir ejecutando en un ciclo (por ejemplo, las operaciones de la `ALU` sobre los registros `A` y `B`), todo lo que conlleve a accesos en memoria necesitaría al menos dos ciclos (uno para acceder a la instrucción y literal en memoria y el otro para acceder a la variable almacenada). Por esto, se asume que para toda instrucción en esta arquitectura se necesitan dos ciclos por lo bajo.

b. **(I2 - II/2014)** Modifique el computador básico, para que este utilice un esquema Von Neumann, *i.e.*, memoria de datos e instrucciones unificadas en una sola.

La principal modificación que se le debe realizar al esquema del computador básico es unir la memoria de instrucciones con la memoria de datos en una sola. Esto claramente modifica los ciclos requeridos por instrucción, por lo que una forma de solucionarlo es dejar por *default* 3 ciclos: El primero se utiliza para enviar el *opcode* y que la unidad de control propague las señales correspondientes (cuidando que no se altere el contenido de los registros), el segundo para enviar el literal correspondiente para realizar las operaciones y el tercero para hacer finalmente la ejecución de la instrucción. Otra consideración importante sería el manejo del `PC`, donde una opción es que estos apunten, dentro de la memoria, a registros de 16 bits (con los primeros 8 bits correspondientes al opcode, y los siguientes al literal), enviando primero los 8 más significativos (instrucción) y luego los 8 menos significativos (literal). Luego, al tercer ciclo se ejecuta la instrucción completa. Otro detalle a cuidar del PC sería que solo fuera alimentado al final del tercer ciclo de toda instrucción, ya que si esto sucediera en el primer o segundo ciclo, habrían inconsistencias en los resultados esperados. Finalmente, faltaría tomar en cuenta que en la ISA se tenga la precaución de que el literal enviado en el segundo ciclo no sea ejecutado en la unidad de control (lo que podría conllevar a problemas en la ejecución de la instrucción final).

Notar que esto es posible hacerlo también en 2 ciclos, asumiendo que de la memoria se extrae el bloque completo que contiene el opcode y el literal y se alcanzan a enviar a los componentes correspondientes antes del flanco de subida (en este caso, `PC` no podría ser alimentado en el primero flanco de subida, solo podría pasar en el segundo).

c. **(I2 - I/2017)** ¿Cuántos ciclos como mínimo puede tomar en un computador con arquitectura Von Neumann, una instrucción que lea y luego modifique el contenido de una posición de memoria?

Antes de desarrollar la pregunta, sin pérdida de generalidad, se asume que **leer** hace referencia a una instrucción del tipo `MOV A,(Dir)`, mientras que **modificar** usa una instrucción del tipo `MOV (Dir),A`, habiendo realizado un cambio sobre el registro donde se almacenó el dato de la memoria.

Aquí nos encontramos con dos casos:

- Asumiendo que no podemos obtener el *opcode* y el literal en un solo ciclo:
  1) Se obtiene el opcode de `MOV A,(Dir)` y se envía a la unidad de control, la que posteriormente propaga las señales de control necesarias para la ejecución.
  2) Se obtiene el literal (dirección de memoria) y se propaga a los multiplexores correspondientes (en este caso, al `Mux Address`).
  3) Se ejecuta la instrucción completa en el flanco de subida del tercer ciclo, donde se obtiene el dato almacenado en `Dir` en el registro `A`.
  4) Se obtiene el *opcode* de la operación a realizar sobre el registro `A` y se envía a la unidad de control, la que posteriormente propaga las señales de control necesarias para la ejecución.
  5) Se obtiene el literal para la operación y se propaga a los multiplexores correspondientes (podemos asumir, sin pérdida de generalidad, un literal `Lit` a sumarse con `A`).
  6) Se ejecuta la instrucción completa en el flanco de subida del sexto ciclo, donde el resultado de la operación se almacena, nuevamente, en el registro `A`.
  7) Se obtiene el *opcode* de `MOV (Dir),A` y se envía a la unidad de control, la que posteriormente propaga las señales de control necesarias para la ejecución.
  8) Se obtiene el literal (dirección de memoria) y se propaga a los multiplexores correspondientes (en este caso, al `Mux Address`).
  9) Se ejecuta la instrucción completa en el flanco de subida del noveno ciclo, donde el dato del registro `A` se almacena en la dirección de memoria `Dir`.
- Asumiendo que sí podemos:
  1) Se obtiene el *opcode* de la instrucción `MOV A,(Dir)` y el literal de la dirección, propagándose a los componentes correspondientes.
  2) Se ejecuta la instrucción, almacenando el dato de la dirección `Dir` en el registro `A`.
  3) Se obtiene el *opcode* de la instrucción a ejecutar sobre el registro `A` y el literal para operar, propagándose a los componentes correspondientes.
  4) Se ejecuta la instrucción, almacenando el resultado de la operación en el registro `A`.
  5) Se obtiene el *opcode* de la instrucción `MOV (Dir),A` y el literal de la dirección, propagándose a los componentes correspondientes.
  6) Se ejecuta la instrucción, almacenando en la dirección `A` el resultado que se encuentra en el registro `A`.

En ambos casos, se ve que la cantidad de ciclos necesaria para la ejecución de lo pedido aumenta considerablemente si se hace el contraste con una arquitectura Harvard.

2. a. **(I2 - II/2014)** Dada la microarquitectura del computador básico, ¿es posible crear una ISA distinta la actual? Argumente su respuesta.

   Sí, es posible, ya que esta se puede modificar de dos formas:

   - Se puede usar un nuevo *opcode* que utilice una combinación de señales no utilizada antes para un nuevo comando. Por ejemplo, la combinación de señales que obtiene la suma del registro `A` y `B` y luego guarda el resultado en ambos.
   - Usando una nueva instrucción que corresponda a la combinación de distintos *opcodes*. Por ejemplo, una instrucción que incremente en una unidad una variable almacenada en una dirección de memoria: `INC (dir) = PUSH A - MOV A,(dir) - INC A - MOV (dir),A - POP A`.

   b. **(I2 - I/2015)** ¿Es posible agregar al Assembly del computador básico la instrucción `MOV A,(A+B)`, sin modificar la microarquitectura? Justifique su respuesta en cualquiera de los dos casos.

   Sí, es posible. Basta con asignarle al nuevo comando los *opcodes* de las siguientes instrucciones existentes de forma consecutiva: `PUSH B - ADD B,A - MOV A,(B) - POP B`. Notar que la primera y última instrucción se usan de forma que no perdamos el valor almacenado en el registro `B`, pues no es el objetivo del comando inicial.

   c. **(I2 - II/2016)** Modifique (solo) la ISA del computador básico para soportar la instrucción `CALL reg`, que permite llamar a la subrutina ubicada en la dirección de memoria almacenada en el registro `reg`.

   Definimos en nuestra ISA la instrucción `CALL reg` de la siguiente forma:

   - Primero se ejecuta la instrucción `MOV (dir),reg` para almacenar en la dirección `dir` el valor almacenado en el registro (ya sea `A` o `B`). Se puede asumir que `dir` es una dirección fija con uso exclusivo para esta instrucción.
   - Añadimos la instrucción `CALL (dir)`, que carga en el registro `PC` el valor almacenado en la dirección `dir`. Notar que esto debe ocurrir en 2 ciclos, ya que primero tenemos que guardar en el stack la posición `PC+1` para poder volver al retornar el llamado y otro para obtener el valor almacenado en `dir` para realizar el salto.

   El cuidado que tendría que tener el programador es de no alterar el contenido de la dirección `dir` en ningún momento.

   d. **(I2 - II/2016)** Implemente, utilizando solo la instrucción `SUBLEQ a,b,c,` la instrucción `SUB a,b`.

   Recordemos que la instrucción `SUBLEQ a,b,c` corresponde a:

   - Mem[a] = Mem[a] - Mem[b]
   - if(Mem[a] ≤ 0) $\rightarrow$ `goto c`.

   Entonces, basta con definir la instrucción `SUB a,b` de la siguiente forma:

   - `dir0: SUBLEQ a,b,dir1`
   - `dir1: ...`

   En `dir1` ya no interesa lo que pase, puse almacenamos en `a` el resultado de `a-b`.