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

IIC2343 – Arquitectura de Computadores (II/2013)

Interrogación 1 - Pauta

Pregunta 1 (2,0 ptos)

El siguiente programa escrito en Java implementa un algoritmo de multiplicación de dos números en complemento a 2. Traduzca el programa al assembly del computador básico, cumpliendo lo siguiente:

- Su programa debe ser una traducción directa del programa en Java, es decir, además de traducir la funcionalidad (multiplicación) debe hacerlo implementando el mismo algoritmo descrito en el programa.
- Su programa debe definir (al menos) las variables `a`, `b` que se ocuparán como operandos de la multiplicación, y la variable `c` que almacenará el resultado.

Pregunta 1.1 (1,0 ptos)

```
public static void main(String[] args)
{
  //Declaracion de variables
  byte a = 12;
  byte b = 3;
  byte c = 0;
  //Loop principal
  while(true)
  {
     //And
     byte d = (byte)(b & 1);
     //If y acumulacion en c
     if(d != 0)
       c += a;
     //Shift right
     b >>= 1;
     //If y break
     if(b == 0)
       break;
     //Shift left
     a <<= 1;
  }
}
```

Solución

```
;Declaracion de variables
DATA:
    a       12
    b       3
    c       0

CODE:

    start:  MOV A, (b)    ;And
            AND A, 1      ;And
            CMP A, 0      ;If y acumulacion c
            JEQ else1     ;If y acumulacion c
            MOV A, (c)    ;If y acumulacion c
            MOV B, (a)    ;If y acumulacion c
            ADD A, B      ;If y acumulacion c
            MOV (c), A    ;If y acumulacion c
    else1:  MOV A, (b)    ;Shift right
            SHR A, A      ;Shift right
            MOV (b), A    ;Shift right
            CMP A, 0      ;If y break
            JNE else2     ;If y break
            JMP end       ;If y break
    else2:  MOV A, (a)    ;Shift left
            SHL A, A      ;Shift left
            MOV (a), A    ;Shift left
            JMP start     ;Loop principal
    end:
```

Puntajes

- Declaracion de variables: 0,1 ptos
- Loop principal: 0,1 ptos
- And: 0,1 ptos
- If y acummulacion c: 0,2 ptos
- Shift right: 0,15 ptos
- If y break: 0,2 ptos
- Shift left: 0,15 ptos

Pregunta 1.2 (1,0 ptos)

Para el siguiente programa escrito en el assembly computador básico, especifique:

- Los valores de los registros `A`, `B`, de las variables `var1`, `var2`, `var3` y del arreglo `arr1` cuando el programa llega a la instrucción asociada al label `label0` (antes que se ejecute esa instrucción).
- Los valores de los registros `A`, `B` y de las variables `var1`, `var2` la **segunda vez** que el programa llega a la instrucción asociada al label `label1` (antes que se ejecute esa instrucción).
- Indique si el programa llega al label `end` o no. En caso de que si llegue especifique los valores de los registros `A`, `B` y de las variables `var1`, `var2` al finalizar el programa.

```
DATA:
    var1    2
    var2    5
    var3    10
    arr1    0
            1
            2

CODE:
            MOV A, (var3)
            MOV B, (var1)
            SUB A, B
            MOV (var2), A
            MOV B, arr1
            MOV A, 2
            ADD A, B
            MOV B, A
            MOV A, (B)
            MOV (var1), A

    label0: MOV A,1
            MOV B,2
    label1: ADD A,1
            ADD B,1
            MOV (var1), B
            MOV (var2), A
            CMP A,B
            JGE label2
            MOV (var1), A
            MOV (var2), B
            JMP label3
    label2: MOV A,4
            CMP A,B
            JLT label1
            ADD B,A
            MOV (var1), A
            MOV (var2), A
            JMP end
    label3: ADD A,B
            JMP label1
    end:
```

Solucion y Puntajes

- **label0:**
  - A = 2 (0,1 ptos)
  - B = arr1 + 2 (0,1 ptos)
  - var1 = 2 (0,025 ptos)
  - var2 = 8 (0,025 ptos)
  - var3 = 10 (0,025 ptos)
  - arr = [0,1,2] (0,025 ptos)
- **label1:**
  - A = 5 (0,1 ptos)
  - B = 3 (0,1 ptos)
  - var1 = 2 (0,05 ptos)
  - var2 = 3 (0,05 ptos)
- **end:**
  - A = 4 (0,1 ptos)
  - B = 8 (0,1 ptos)
  - var1 = 4 (0,1 ptos)
  - var2 = 4 (0,1 ptos)

Pregunta 2 (2,0 ptos)

Pregunta 2.1 (1,0 ptos)

Para un determinado programa la *secuencia efectiva* de instrucciones corresponde a la secuencia de todas las instrucciones que se ejecutaron desde que comenzó hasta que terminó el programa, incluyendo las instrucciones que se repitieron por saltos o llamados a subrutinas.

A modo de ejemplo, para el siguiente programa:

```
CODE:
            MOV A, 0    //Instruccion 0
            MOV B, 2    //Instruccion 1
label1:     CMP A,B     //Instruccion 2
            JGE end     //Instruccion 3
            ADD A,1     //Instruccion 4
            JMP label1  //Instruccion 5
end:
```

La secuencia efectiva de instrucciones es:

```
MOV A, 0    //Instruccion 0
MOV B, 2    //Instruccion 1
CMP A,B     //Instruccion 2
JGE end     //Instruccion 3
ADD A,1     //Instruccion 4
JMP label1  //Instruccion 5
CMP A,B     //Instruccion 2
JGE end     //Instruccion 3
ADD A,1     //Instruccion 4
JMP label1  //Instruccion 5
CMP A,B     //Instruccion 2
JGE end     //Instruccion 3
```

Realice las modificaciones necesarias al computador básico de manera que se pueda implementar la instrucción `NUM_INSTRUCTIONS (var1)` la cual almacena en la variable `var1` la cantidad de instrucciones que efectivamente se ejecutaron hasta que se llamó a esta instrucción (no incluyendo a esta instrucción).

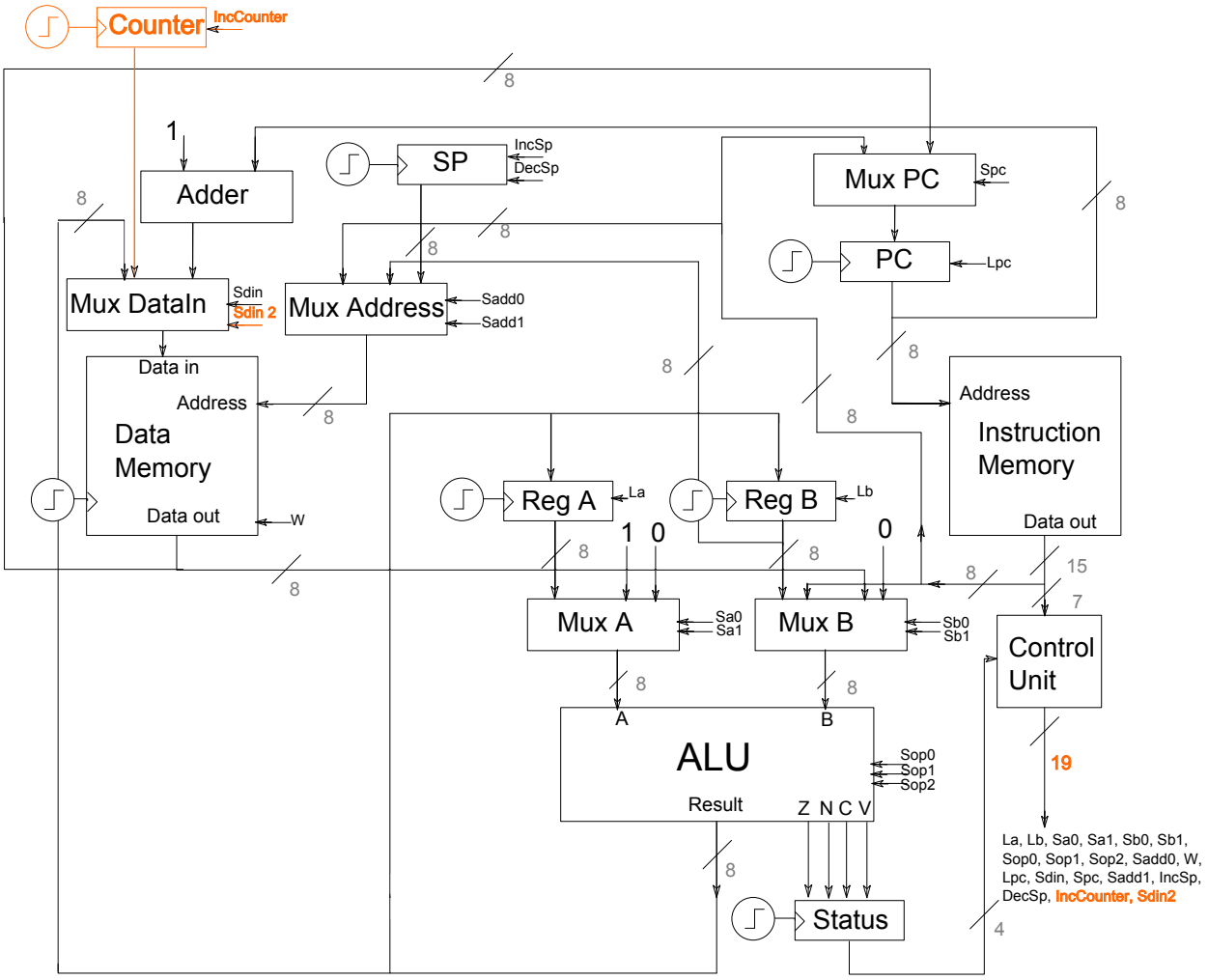
A modo de ejemplo, para el programa anterior, si se llama a esta instrucción luego del label `end` se debería almacenar el valor 12 en la variable `var1` dado que se ejecutaron 12 instrucciones.

Importante 1: Para esta pregunta puede asumir que todas las instrucciones se ejecutan en un ciclo del clock.

Importante 2: No puede agregar memorias para resolver esta pregunta.

Solución

Una posible solución es la siguiente:



- Se agrega un contador al computador con una señal de control IncCounter, que hace que el contador se increment si está habilitada.
- Se aumenta el tamaño del Mux de data in de la memoria de datos, y se agrega la salida del contador como entrada posible de este Mux.
- A todas las instrucciones se les agrega la señal de control IncCounter con valor 1, para que se incremente el contador luego de cada llamado, exceptuando a la nueva instrucción `NUM_INSTRUCTIONS (var1)` la cual no activa esta señal.
- Las señales de control relevantes de la nueva instrucción `NUM_INSTRUCTIONS (var1)` son:
  - La señal de selección del MUX Data In se elige para que se escoja el valor del contador como entrada a la memoria (Sdin = 10 por ejemplo).
  - Como dirección para la memoria de datos se elige Saddress = Literal.
  - La señal IncCounter se setea en 0.
  - Todas las otras señales de Load o Incremento/Decremento se setean en 0.

Puntaje

- Lograr que de alguna forma se cuenten todas las instrucciones que se ejecutaron, ya sea a medida que se ejecutan o después. No es válido ocupar el Program Counter, porque este solo incrementa si no hay instrucciones de salto, si las hay no tendrá almacenado el número total de instrucciones (0,5 ptos).
- Lograr que la instrucción `NUM_INSTRUCTIONS (var1)` configure bien la memoria:
  - Agrandar mux de data in para que acepte una entrada más, y seleccionar esta entrada que debería tener el número de instrucciones. (0,2 ptos)
  - Setear el mux de address para que se elija como dirección el literal de la instrucción. (0,1 ptos)
  - Setear la memoria en modo escritura (W = 1). (0,1 ptos)
  - Asegurarse que todo el resto de lase señales de carga esten en 0 (LoadA, LoadB, LoadPC, IncSP, DecSP).(0,1 ptos)
- Descuento si las modificaciones rompen alguna de las instrucciones existentes (-0,3 ptos).

Pregunta 2.2 (1,0 ptos)

Uno de los modos de direccionamiento existentes en el computador básico es el **direccionamiento directo** el cual, por ejemplo, se ocupa al usar la instrucción `MOV A, (var1)`. Esta instrucción carga en el registro A el valor que **directamente** está ubicado en la dirección de memoria datos asociada a `var1`. A modo de ejemplo, en la tabla de memoria que se encuentra más abajo, si la dirección asociada a `var1` fuese 2, al ejecutar la instrucción `MOV A, (var1)`, el registro A quedaría con el valor 5, o sea el valor de la palabra de memoria ubicado en la dirección 2.

Dirección	Palabra
0	5
1	4
2	5
3	0
4	0
5	7
6	1
7	1

Se desea agregar al computador básico la capacidad de realizar **direccionamiento indirecto**, en particular usted debe permitir que el computador básico soporte la instrucción `MOV A, ((var1))`. Esta instrucción realiza lo siguiente:

1. Obtiene la palabra almacenada en la dirección asociada a `var1`. Siguiendo el ejemplo anterior, suponiendo que la dirección asociada a `var1` es 2, el valor que se obtiene sería 5
2. Ocupa este valor obtenido para **volver a direccionar la memoria de datos**, y obtener una nueva palabra de memoria. En el ejemplo el valor 5 se ocupa como dirección, y se obtiene el valor que se encuentra en la dirección 5 de la memoria, o sea 7.

3. Se almacena este último valor en el registro A. En el ejemplo A = 7.

Solución

Una posible solución es implementar esta instrucción sin modificar el hardware. La siguiente secuencia de instrucciones implementa la instrucción:

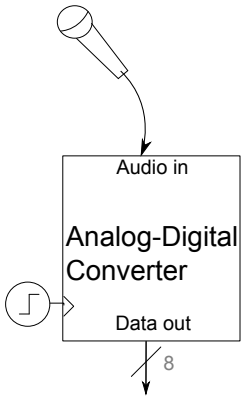
```
PUSH B
MOV B, (var1)
MOV A, (B)
POP B
```

Puntaje

- Lograr la funcionalidad asociada a la instrucción (direccionamiento indirecto), ya sea sin modificaciones al computador, o agregando alguna modificación que sea util (1,0 ptos).
- Descuento si se implementa por software, pero no se conserva el estado del computador, por ejemplo si se ocupa el registro B, pero no se hace un backup y restoración previa, o si se ocupa la memoria de datos para escribir algun valor potencialmente borrando lo que había antes (-0.5 ptos).
- Descuento si las modificaciones rompen alguna de las instrucciones existentes (-0,3 ptos).

Pregunta 3 (2,0 ptos)

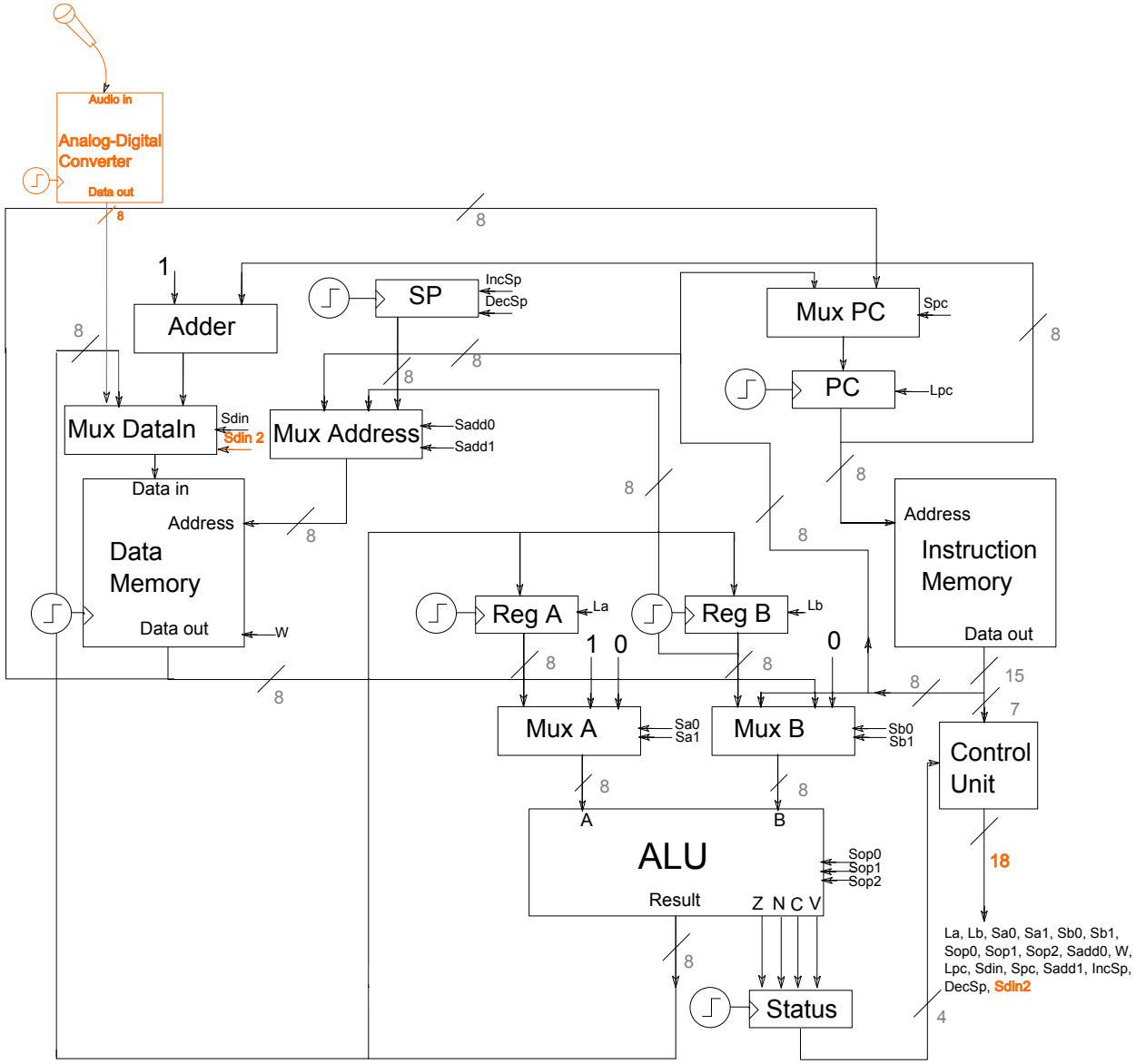
Se requiere agregar al computador básico la capacidad de procesar y almacenar audio desde un micrófono. Para esto se cuenta con un conversor análogo-digital, el cual recibe como entrada una señal continua que representa las variaciones de voltaje captadas por el micrófono y entrega como salida muestras de 8 bits cada cierto tiempo T (período de muestreo), de manera automática. El conversor está contectado al mismo clock del computador básico y se sabe que el período de muestreo T es 8 veces el tiempo que toma un ciclo del clock, lo que permite mantener sincronización. A continuación se muestra el diagrama del conversor análogo-digital:



1. Modifique el computador básico agregando un conversor análogo-digital de manera de que permita almacenar la muestra en memoria. Para esto debe agregar la instruccion `READ_ADC (B)`, la cual almacena una muestra del conversor análogo-digital, en la dirección de memoria de datos apuntada por el valor almacenado en el registro `B`. Conecte el conversor al computador básico de manera de que se pueda ejecutar esta instrucción e indique que señales de control se deben activar para ejecutar la instrucción.

2. Escriba un programa en el assembly del computador básico que implemente un loop infinito que vaya almacenando de manera sucesiva las muestras que van llegando desde el conversor análogo-digital en el arreglo almacenado en la dirección memoria apuntada por el label `vector`. Asegurese de obtener todas las muestras y no repetir muestras.

Solución



La instrucción `READ_ADC (B)` configura la memoria para que la entrada de datos sea el ADC y la dirección el registro B.

```
        MOV B, vector    ; Inicializar B con posicion inicial del arreglo vector
loop:   READ_ADC (B)     ; Leer muestra
        INC B            ; Incrementar en 1 posicion del vector
        CMP A,B          ; Ejecutar 5 instrucciones que no hagan nada util para
        CMP A,B          ; sincronizar programa con el periodo de muestreo
        CMP A,B
        CMP A,B
        CMP A,B
        JMP loop         ; Loop inifinto
```

Puntaje

- Conectar ADC a mux de entrada de datos de la memoria (0.5 ptos).
- Lograr que la instrucción `READ_ADC (B)` configure bien la memoria:
  - Agrandar mux de data in para que acepte una entrada más, y seleccionar esta entrada que debería tener el ADC. (0,2 ptos)
  - Setear el mux de address para que se elija como dirección el registro B. (0,1 ptos)
  - Setear la memoria en modo escritura (W = 1). (0,1 ptos)
  - Asegurarse que todo el resto de lase señales de carga esten en 0 (LoadA, LoadB, LoadPC, IncSP, DecSP).(0,1 ptos)
- El programa que lee las muestras debe cumplir con:
  - Inicializar B con posicion inicial del arreglo vector (0.1 ptos)
  - Leer muestra llamando a instrucción `READ_ADC (B)` (0.2 ptos)
  - Incrementar en 1 posicion del vector (0.1 ptos)
  - Ejecutar 5 instrucciones que no hagan nada util para sincronizar programa con el periodo de muestreo (0.5 ptos)
  - Loop inifinto (0.1 ptos)
- Descuento si las modificaciones rompen alguna de las instrucciones existentes (-0,3 ptos).