Tarea 6

Arquitectura de Computadores – IIC2343

Rafael Fernández

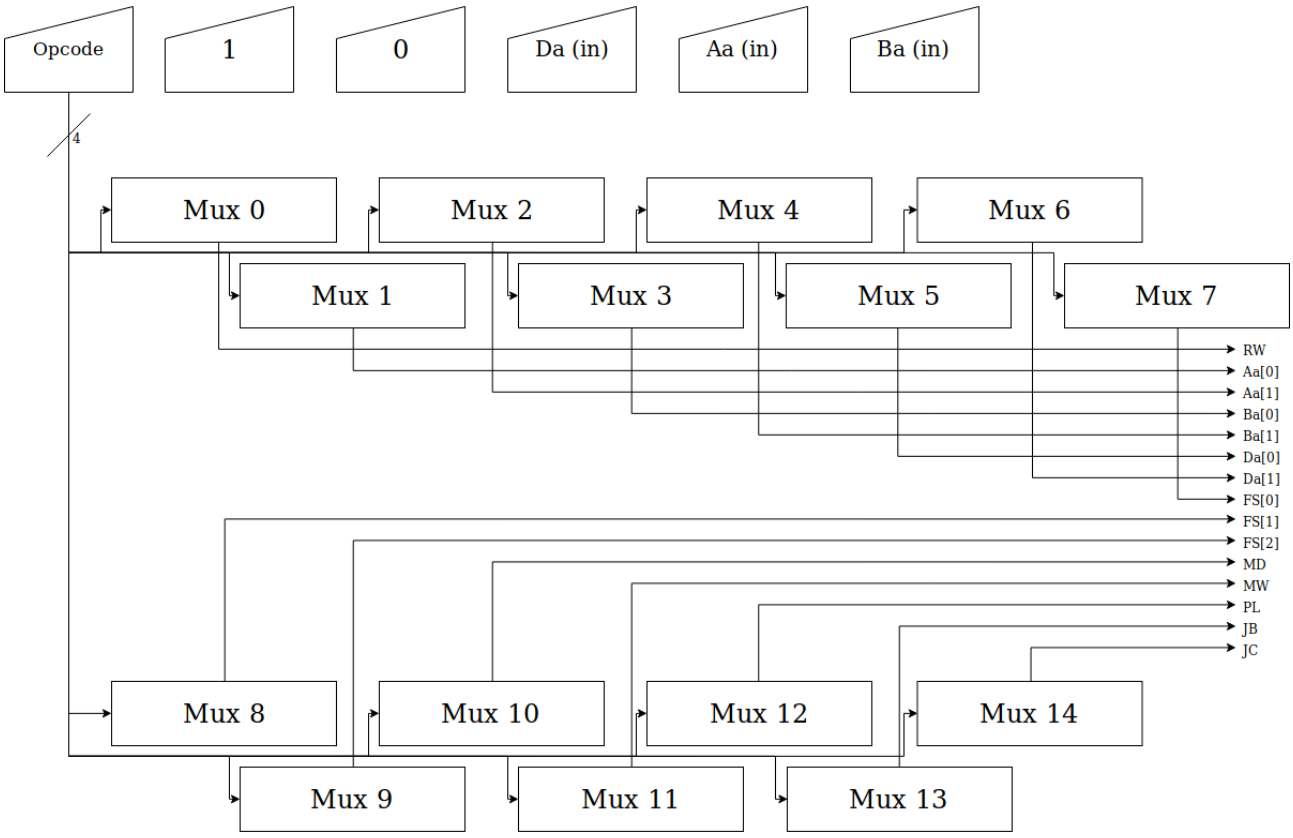
Pregunta 1)

A continuación se muestran los valores que debiesen tomar las señales de control para ejecutar cada una de las instrucciones asociadas a los opcodes dados.

Para las señales de control indicadas con “arg”, se refiere a que existe una instrucción para cada una de las distintas combinaciones entre ese opcode y los posibles bits para la señal de control marcada.

Opcode	INS	RW	Aa	Ba	Da	FS	MD	MW	PL	JB	BC
0000	MOVA	1	arg	00	arg	000	0	0	0	0	0
0001	ADD	1	arg	arg	arg	001	0	0	0	0	0
0010	SUB	1	arg	arg	arg	010	0	0	0	0	0
0011	AND	1	arg	arg	arg	011	0	0	0	0	0
0100	OR	1	arg	arg	arg	100	0	0	0	0	0
0101	XOR	1	arg	arg	arg	101	0	0	0	0	0
0110	NOT	1	arg	00	arg	110	0	0	0	0	0
0111	MOVB	1	00	arg	arg	111	0	0	0	0	0
1000	LD	1	arg	00	arg	000	1	0	0	0	0
1001	ST	0	arg	arg	00	000	0	1	0	0	0
1010	BRZ	0	arg	00	00	000	0	0	1	0	0
1011	BRN	0	arg	00	00	000	0	0	1	0	1
1100	JMP	0	arg	00	00	000	0	0	1	1	0

Diagrama de Instruction Decoder:



Para no complicar demasiado el diagrama, se omitieron las conexiones de entrada de los multiplexores, las cuales serán explicadas a continuación:

Cada multiplexor entrega como output un bit de señal de control. Para esto, cada uno tiene trece entradas conectadas a los inputs (Trapecios en la parte superior del diagrama) correspondientes a la n-esima instrucción. Cada una de estas entradas se selecciona con el opcode.
De esta forma, con el opcode 0000, por ejemplo, cada multiplexor elegiría su primera entrada, resultando en una señal de control de 16 bits que corresponde con la instrucción MOVA.

Para ilustrar las conexiones de los multiplexores, se mostrarán a continuación las tablas de verdad de dos multiplexores. El resto de los multiplexores siguen la misma lógica.

MUX 0	
Input	**Output**
0000	1
0001	1
0010	1
0011	1
0100	1
0101	1
0110	1
0111	1
1000	1
1001	0
1010	0
1011	0
1100	0

MUX 2	
Input	**Output**
0000	00
0001	Ba
0010	Ba
0011	Ba
0100	Ba
0101	Ba
0110	00
0111	Ba
1000	00
1001	Ba
1010	00
1011	00
1100	00

Cabe notar que este diseño es simple, pero bastante ineficiente.

Pregunta 2)

1.

Cada instrucción mostrada a continuación es independiente de la otra, por lo que, con la excepción de las instrucciones SHR y SHL, los cambios en el computador NO se acumulan para las instrucciones suguientes.

INC: Para soportar esta instrucción, se podría agregar un multiplexor de dos entradas entre la conexión de la salida B del Register File y la Functional Unit. A este nuevo multiplexor (Llamémosle MUX B) se le conectaría en una entrada la salida B del Register File y en otra entrada una señal que esté siempre activa de un bit. Para que el computador haga uso de su nuevo hardware, habría que agregar una nueva señal de control “MB” de un bit, que controle la salida del MUX B, eligiendo con cero la entrada de B y con uno la señal activa (que representa 1). Como todas las instrucciones previas no requieren sumar uno, se tendría que stablecer la señal de control MB en cero para todas ellas.

Para Ejecutar la instrucción usariamos, entonces, las siguientes señales de control:

INS	RW	Aa	Ba	Da	FS	MD	MW	PL	JB	BC	MB
INC	1	arg	00	arg	001	0	0	0	0	0	1

DEC: Para esta instrucción el análisis es análogo al anterior, pero en vez de elegir la operación de suma en la ALU, tenemos que elegir la de resta. Esta se selecciona estableciendo la señal de control FS en 010. El resto de lo dicho para la instrucción INC aplica acá también.
Las señales de control serían:

INS	RW	Aa	Ba	Da	FS	MD	MW	PL	JB	BC	MB
DEC	1	arg	00	arg	010	0	0	0	0	0	1

SHR: Una forma de implementar esta instrucción sería agregando a la Functional Unit un circuito de Shfit Right como el visto en el Control 1. El output de este circuito debe poder ser seleccionado por la señal de control FS, pero como ya se han asignado ocho operaciones y se tienen tres bits, sería necesario agregarle un bit extra a la señal de control FS para soportar la selección de más opciones. Para que el resto del computador siga funcionando, habría que agregarle un cero a la izquierda a todas las señales de control FS.
Las señales de control serían:

INS	RW	Aa	Ba	Da	FS	MD	MW	PL	JB	BC
SHR	1	00	arg	arg	1000	0	0	0	0	0

SHL: El argumento es exactamente igual al anterior, excepto por que, obviamente, el circuito de la Functional Unit debe ser de Shift Left en vez de Shift Right (También visto en el control 1). La señal de control FS en este caso sería 1001, ya que asumimos que 1000 está ocupado con la salida del circuito de Shift Right.
Las señales de control serían:

INS	RW	Aa	Ba	Da	FS	MD	MW	PL	JB	BC
SHL	1	00	arg	arg	1000	0	0	0	0	0

LDI: Esta es la instrucción que requiere más cambios. Primero, la Instruction Memory ahora debe dar como output los bits mostrados en el enunciado más una cantidad de bits que representen un posible literal. La cantidad dependerá del tamaño de datos que puede manejar el computador, determinado por los registros y la memoria, generalmente. El bus correspondiente a los bits de salida mostrados en el enunciado seguirán conectados al Instruction Decoder, mientras que el bus del literal será conectado a una entrada de un nuevo multiplexor “MUX A”, cuyo output será conectado a la entrada Address de la Data Memory. También la previa conexión que había entre al salida A del Register File y esa entrada de la Data Memory, ahora será entre la A del register File y la otra entrada del nuevo MUX A. Para controlar entre las dos salidas de este multiplexor, es necesario agregar una señal de control MA, la cual seleccione con cero la salida de A del Register File y con uno la del literal de la Instruction Memory. Esta señal de instrucción se debe establecer en cero para que el resto de las instrucciones sigan funcionando. Además, como el formato de cada instrucción cambió, es necesario

establecer en cero (aunque en realidad puede ser cualquier valor) todos los bits de literal en las demás instrucciones.
Las señales de control para LDI serían:

INS	RW	Aa	Ba	Da	FS	MD	MW	PL	JB	BC	MA
LDI	1	00	00	arg	000	1	0	0	0	0	1

ADI: Para esta instrucción habría que hacer el mismo cambio de formato de instrucciones (agregar la posibilidad de entregar un literal) que en la instrucción LDI. Luego, hay que agregar un multiplexor "MUX B2" (para evitar confusiones con MUX B), el cual tenga su output conectado a la entrada B de la Functional Unit y que en una de sus entradas esté conectada la salida B del Register File y en otra el bus proveniente del literal de la memoria de instrucciones. Luego, este multiplexor se controlaría con una señal MB2, que seleccione con un cero la salida B del Register File y con un uno el literal. Para que el resto de las instrucciones (las de la pregunta 1) sigan funcionando, habría que establecer en todas ellas un valor cero para MB2.
Las señales de control para esta instrucción serían:

INS	RW	Aa	Ba	Da	FS	MD	MW	PL	JB	BC	MB2
ADI	1	arg	00	arg	001	0	0	0	0	0	1

2.

Con los nuevos cambios, particularmente con los introducidos al implementar la instrucción LDI, es posible instroducir la instruccion **STD** (Store Dir), la cual, al momento de ser ejecutada, ejecutaría la siguiente operación M[Lit] = R[Ba]. Esto quiere decir que ahora el computador puede guardar el valor de un registro en una dirección de memoria correspondiente al literal pasado a la instrucción.
Es posible la implementación de esta instrucción ya que el MUX A permitiría ahora a la memoria obtener como dirección el literal de la Instruction Memory, dando más libertades respecto al método anterior, el cual obtenía la dirección a partir de la salida A de la Register File.

Pregunta 3)

1.

Para poder comunicarse con dispositivos mapeados a memoria, hay que instalar, en primer lugar una pieza de hardware denominada "Address Decoder". Esta pieza se instala en el address bus que comunica la CPU con los distintos dispositvos de I/O, y su función es transmitir la información por el bus hacia los dispositivos de I/O solo en el caso de que contengan una dirección de memoria que se encuentre en el espacio asignado a los dispositivos. De esta manera, y gracias a que algunas direcciones de memoria ahora hacen referncia a dispositivos externos, podemos utilizar las instrucciones de assembly MOV para comunicarnos con los dispositivos. Estas

direcciones nos permitirán leer o escribir datos de los registros o buffers de los controladores de los dispositivos, permitiendo así la comunicación.
Para el caso en que queremos transferir datos a un dispositivo mapeado a memoria, habría que ejecutar una instrucción como MOV (dir), A. Lo que ocurriría en el computador sería lo siguiente: La CPU pondría el literal pasado a la instrucción en el bus de direcciones. Luego, el Address Decoder detecta que la dirección corresponde a un dispositivo y la transmite por el bus hasta el dispositivo de I/O que corresponde. Después, la CPU pondría el valor del registro A en el Data Bus, para que así el dispositivo de I/O pueda acceder al valor entregado. Finalmente, el controlador del dispositivo guardaría en el buffer, en la dirección indicada, el valor disponible en el Data Bus.
Si queremos leer información del dispositivo, el proceso es muy similar. Para esto, utilizaríamos una instrucción como MOV A, (dir), la cual permitiría obtener en el registro A la información del dispositivo. Los primeros pasos son iguales a lo explicado anteriormente, pero ahora quien entrega información al Data Bus es el controlador del dispositivo, y quien la recibe y actualiza sus valores es la CPU.

2.

Debido a que las direcciones que podemos acceder en los dispositivos de I/O estan limitadas por el espacio de direcciones que se ha reservado para estos dispositivos, es preferible utilizar dispositivos de I/O que tengan un buffer relativamente pequeño, de modo que todas sus direcciones sean accesibles mediante (ojalá una porción) el espacio de direcciones asignado. Si se tiene un dispositivo que produce muchos datos y los guarda en su buffer, como por ejemplo, una cámara o una tarjeta de video, entonces no conviene utilizarlo con memory mapping ya que nos ocuparía prácticamente todas las direcciones o incluso no nos alcanzarían para acceder a todos los datos del dispositvo.

3.

Generalmente se prefiren las interrupciones antes que el polling debido a que el primero es mucho más eficiente. Cuando se hace polling, se usan muchos ciclos de la CPU para detectar cambios en el dispositivo y así ejecutar una operación determinada. Con interrumpuciones, en cambio, el dispositivo es quien avisa a la CPU que ha ocurrido y que debe actuar frente a eso. Un ejemplo de esto, sería por ejemplo una cámara de fotos conectada a un computador. Se quiere que cuando se presione el botón para sacar fotos, se guarde en la memoria del computador una nueva fotografía. Si se hiciera uso de polling, la CPU debería invertir muchos ciclos revisando continuamente si se ha presionado el botón, causando que otras operaciones demoren más en ejecutarse. En cambio, con interrumpciones, en el momento en que el usuario presiona el botón de la cámara, esta le avisa al computador y se procesa en ese momento la foto, ahorrando así muchos ciclos.

4.

La función del vector de interrupciones es la de almacernar las direcciones donde se encuentran las Interrumpt Service Routines (ISRs) asociadas a cada uno de los dispostivos, de forma que cuando uno de estos genere un interrumpt request, se pueda acceder al vector de interrupciones y saber a que dirección de memoria saltar para ejecutar la subrutina asociada al dispositivo que realizó la interrupción.

5.

La principal diferencia entre ambos tipos de interrumpciones es quien gatilla una ISR. En el caso de las interrumpciones por hardware, es un dispositivo físico quien genera una IRQ y luego la CPU ejecuta una ISR asociada a ese dispositivo. Por otra parte, en las interrupciones de software, es en un programa que está siendo ejecutado donde se gatilla la ejecución de una ISR. El autor es un programador, no un dispositivo físico. Cabe mencionar que la interrupción por software no deshabilita automáticamente la atención a otras interrupciones, por lo que hay que hacerlo manualmente mediante el programa (cambiar el Interrumption Flag).
Un ejemplo de una interrumpción por hardware sería una causada por el movimiento del mouse, de modo que este le avisa al computador cada vez que pasa algo en vez de que el computador revise continuamente.
Por otra parte, un ejemplo de una interrumpción por software podría ser la de la comunicación de un programa con la tarjeta de video, donde el primero, en alguna línea de su código modifica variables que generan cambios en la pantalla y luego gatilla una interrumpción por software para que se ejecute una ISR que se comunique con la tarjeta de memoria y le envíe las nuevas variables para que esta haga los cambios en la pantalla.

Pregunta 4)

1.

A continuación se elegirán direcciones de memoria arbitrarias para asociar los componentes del controlador del robot.

Dirección de memoria	Componente
0x0	Registro de estado (OUT)
0x1	Registro de comandos (IN)

Cabe mencionar que también se debe definir una ISR para manejar las interrupciones del robot y el vector de interrumpciones debe guardar la dirección de memoria donde comienza la ISR.

2.

Comandos

Acción	Valor
Encender	000
Apagar	001
Rotar	010
Avanzar	011
Examinar al frente	100

Estados

Estado	Valor
Encendido	000
Avanzando	001
Rotando	010
Muralla al frente	011
Libre al frente	100

Pregunta 5)

Al estar ejecutándose muchos procesos simultáneneamente, el procesador, para generar la ilusión de paralelismo, debe estar cambiando constantemente entre la ejecución de cada uno de los procesos. Esto involucra interrumpir el proceso actual, guardar el estado en el PCB, cambiar de proceso, modificar el PTBR, rellenar la tabla de páginas del siguiente proceso si es necesario y restaurar el estado de los registros con el PCB del siguiente programa, etc. Todas estas operaciones son costosas, por lo que un computador que está ejecutando muchos procesos simultáneamente, pasará demasiados ciclos haciendo el cambio de contexto, generando pérdidas de rendimiento. Esto es desde el punta de vista de la CPU, pero también hay desventajas por el lado de la memoria. Si existen muchos procesos en ejecución, la memoria, por limitaciones de espacio de almacenamiento, no puede ser mapeada simultáneamente a las páginas virtuales de cada uno de los procesos, por lo que ocurren repetidamente los procesos de swap in y swap out, donde se llevan y traen marcos físicos al disco duro. Este tipo de memorias secundarias son muy lentas en comparación con las otras memorias, por lo que un acceso repetido a esta pieza de hardware se traduce en una ejecución lenta de los procesos. Otro problema que puede ocurrir, pero que es de menor impacto, es el hecho de que muchos procesos ejecutándose aumentará la cantidad de memoria utilizada en tablas de páginas, por lo que si la memoria es pequeña, pueden haber problemas de almacenamiento.
Para solucionar la disminución de rendimiento por el continuo acceso a disco duro (o memoria secundaria), se debe aumentar el tamaño de la memoria principal, de modo que más mapeos de páginas virtuales con marcos físicos puedan estar presentes de manera simultánea y así se evite tener que recurrir al disco duro.