Danna Guevara Quesada C23562

UNIVERSIDAD DE COSTA RICA
ESCUELA DE INGENIERÍA ELÉCTRICA
MICROPROCESADORES
IE0623

EIE
Escuela de
Ingeniería Eléctrica

TAREA #3

En un sistema de control por IoT se reciben datos a 12 bits que contienen patrones de las salidas del controlador que se deben activar/desactivar, lo patrones recibidos son procesados en bloques de hasta 50 datos, este valor estará almacenado en una constante denominada CANT. Los datos son recibidos por medio de una conexión Wi-Fi y trasladados a una tabla en la memoria del procesador denominada DATOS_IoT, todos los datos se almacenan con la misma longitud. Como es ampliamente conocido los datos transferidos por internet están codificados en ASCII y así se almacenan en esta tabla. El procesador debe convertir los Datos_IoT a binario y ponerlos en un arreglo DATOS_BIN. Luego de esto los valores en DATOS_BIN se deben transferir separados en nibbles a tres arreglos denominados Nibble_UP, Nibble_MED y Nibble_LOW, para ser transmitidos a las unidades remotas de entrada/salida (RIO). Cada nibble se almacenará en un byte separado. De esta manera, por ejemplo, si se recibe en Datos_IoT el valor ASCII (codificado en HEX): $32$37$35$35 entonces el patrón resultante a almacenar en DATOS_BIN es 101011000011 al realizar su conversión, siendo que los bits que están en 1 son las seis salidas a activar y las 6 salidas a desactivar, en este caso. Consecuentemente en Nibble_UP se colocará un valor $0A, en Nibble_MED un valor $0C y en Nibble_LOW un valor $03.

Se debe diseñar e implementar un programa para realizar estas acciones en la tarjeta DRAGON 12+. Este programa debe incluir únicamente las siguientes subrutinas:

Subrutina GET_CANT: Esta subrutina es invocada desde el programa principal y debe recibir el valor de CANT por medio del teclado, utilizando la subrutina GETCHAR. La subrutina debe validar que el valor ingresado sea un número válido en el rango permitido. La subrutina GET_CANT debe desplegar el siguiente mensaje en pantalla:

> INGRESE EL VALOR DE CANT (ENTRE 1 Y 50):

Luego de que se presione una tecla la subrutina debe validar que la tecla presionada es numérica y está en el rango permitido, en caso contrario debe permanecer leyendo el teclado hasta que una tecla numérica válida sea presionada. Luego de leer el primer dígito la subrutina GET_CANT, desplegará ese valor en la pantalla y procederá a leer la segunda tecla. Al recibirse la segunda tecla numérica válida, la subrutina debe calcular el valor ingresado como un binario sin signo y devolver el valor en la variable CANT. El valor ingresado por el usuario debe aparecer al final del mensaje en el Terminal. Los valores entre 1 y 9 deberán ser recibidos como 01, 02…, 09. El valor 00 NO es un valor válido y no debe ser aceptado. La subrutina no retornará hasta que un valor válido haya sido ingresado y quede en CANT. Las únicas estructuras de datos que esta subrutina debe usar son la variable CANT y el mensaje ASCII a colocar en pantalla.

Subrutina ASCII_BIN: Esta subrutina será la encargada de tomar cada valor ASCII ubicado en Datos_IoT convertirlo a binario y poner el resultado en un arreglo DATOS_BIN. La subrutina recibe del programa principal las direcciones Datos_IoT y Datos_BIN por medio de la pila y recibe por memoria el valor de la cantidad de datos a procesar (CANT). Para ello el programa principal debe apilar DATOS_BIN y Datos_IoT, en ese orden, antes de llamar la subrutina, debe usar Load Effective Address administrar los punteros en la pila. Además de las estructuras de datos indicadas, esta subrutina únicamente utilizará CONT, OFFSET y ACC. Para barrer los arreglos se debe utilizar direccionamiento indexado por acumulador.

Subrutina Mover: Esta subrutina es la encargada de separar los patrones de 12 bits en sus 3 "nibbles" constitutivos, colocando cada parte en los arreglos definidos para tal fin. Los "nibbles" deben ser colocados en el mismo orden en que aparecen en Datos_BIN. La subrutina recibe la dirección Datos_BIN por medio del registro índice X y las direcciones de Nibble_UP, Nibble_MED y Nibble_LOW por medio de tres punteros (variables) ubicadas en las direcciones con el mismo nombre de los arreglos. Además recibe por memoria el valor de la cantidad de datos a procesar (CANT).

Subrutina IMPRIMIR: Esta subrutina es llamada desde el programa principal y debe imprimir los resultados de la siguiente manera:

>CANTIDAD DE VALORES PROCESADOS: <CONT>

Este resultado debe aparecer dos líneas por debajo del último mensaje desplegado.

Adicionalmente esta subrutina deberá imprimir, en el terminal, los arreglos Nibble_UP, Nibble_MED y Nibble_LOW calculados, de la siguiente manera.

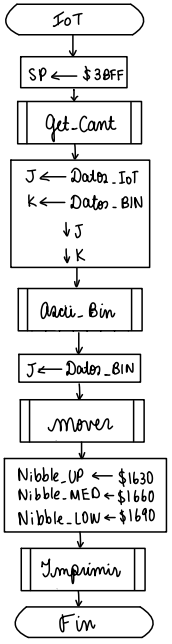
> Nibble_UP: <nibbleUp_1> ,<nibbleUp_2>, …, <nibbleUp_k>

> Nibble_MED: <nibbleMed_1> ,<nibbleMed_2>, …, <nibbleMed_k>

> Nibble_LOW: <nibbleLow_1> ,<nibbleLow_2>, …, <nibbleLow_k>

Note que los datos de los arreglos deben imprimirse separados por una coma, excepto el último. Estos resultados deberán aparecer dos líneas por debajo del último mensaje desplegado y separados dos líneas entre sí. Los datos de los arreglos deben imprimirse en hexadecimal. En estas subrutinas solo pueden usarse las estructuras de datos indicadas además de CANT.

Nota: Para desplegar todos los mensajes se debe utilizar la subrutina Printf.



Estructuras de datos:

Datos_IoT: constante tipo word, dirección de la tabla de datos

Datos_BIN: constante tipo word, dirección del array

Nibble_UP: constante tipo word, puntero apunta dirección Nibble_UP

Nibble_MED: constante tipo word, puntero apunta dirección Nibble_MED

Nibble_LOW: constante tipo word, puntero apunta dirección Nibble_LOW

Subrutina GET_CANT: Esta subrutina es invocada desde el programa principal y debe recibir el valor de CANT por medio del teclado, utilizando la subrutina GETCHAR. La subrutina debe validar que el valor ingresado sea un número válido en el rango permitido. La subrutina GET_CANT debe desplegar el siguiente mensaje en pantalla:

> INGRESE EL VALOR DE CANT (ENTRE 1 Y 50):

Luego de que se presione una tecla la subrutina debe validar que la tecla presionada es numérica y está en el rango permitido, en caso contrario debe permanecer leyendo el teclado hasta que una tecla numérica válida sea presionada. Luego de leer el primer dígito la subrutina GET_CANT, desplegará ese valor en la pantalla y procederá a leer la segunda tecla. Al recibirse la segunda tecla numérica válida, la subrutina debe calcular el valor ingresado como un binario sin signo y devolver el valor en la variable CANT. El valor ingresado por el usuario debe aparecer al final del mensaje en el Terminal. Los valores entre 1 y 9 deberán ser recibidos como 01, 02..., 09. El valor 00 NO es un valor válido y no debe ser aceptado. La subrutina no retornará hasta que un valor válido haya sido ingresado y quede en CANT. Las únicas estructuras de datos que esta subrutina debe usar son la variable CANT y el mensaje ASCII a colocar en pantalla.

① Definir las estructuras

② Imprimir el mensaje

③ Leer la primera tecla

↳ si es inválida osea menor que 0 o mayor que 5, se vuelve a ③

④ se evalúa la primera tecla

↳ si es 0 se va a ⑤→

↳ si es 1 - 4 se va a ⑤

↳ si es 5 se va ⑤

⑤ Leer la segunda tecla

↳ si es inválida osea menor que 0 o mayor que 5, se vuelve a ⑤

⑥ se evalúa la segunda tecla

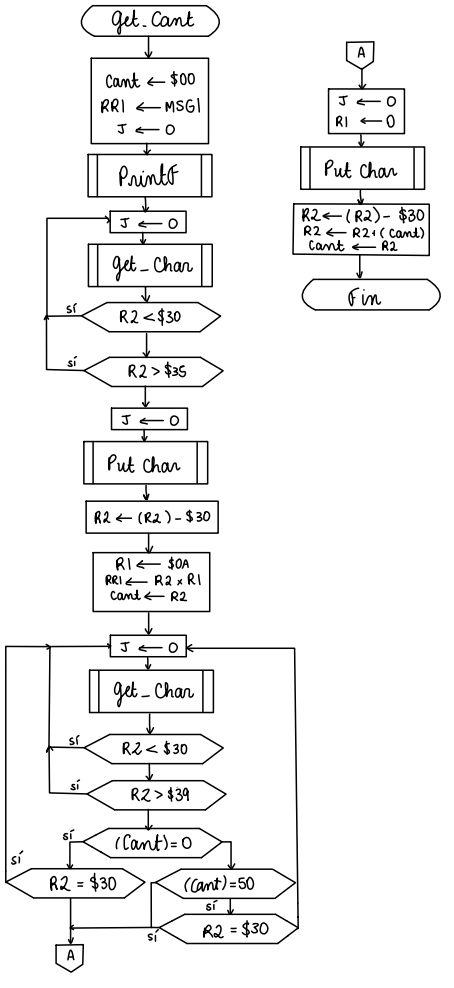
↳ si la primera = 0, esta debe ser de 1 - 9, si es 0 ③

↳ si la primera = 1 - 4, puede ir de 0 - 9

↳ si la primera = 5, esta debe ser 0, si es 1 - 9 ③

⑦ Se guarda en cant como binario sin signo

⑧ Debe de mostrar el valor



MSG1: Ingrese el valor de la variable cant (entre 0 y 50):

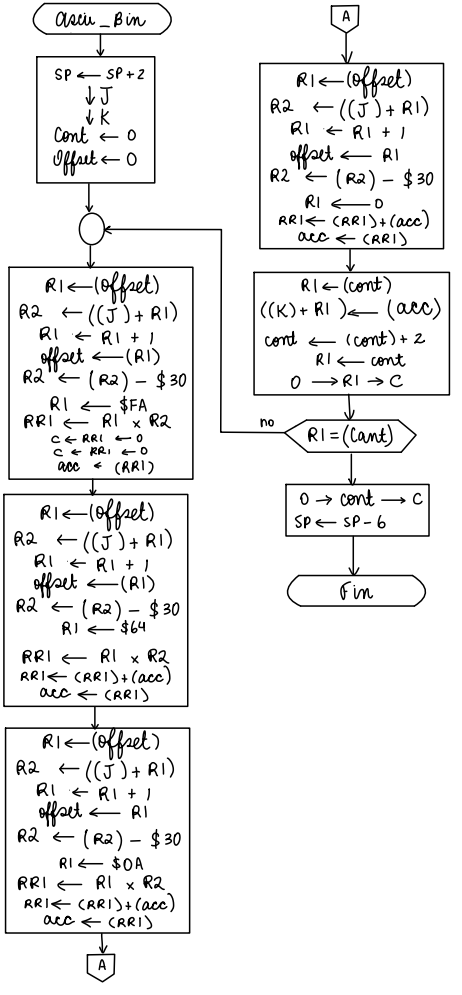
Estructuras de datos:

MSG1: cadena de caracteres

cant: variable tipo byte donde se guarda la cantidad de datos a procesar

Subrutina ASCII_BIN: Esta subrutina será la encargada de tomar cada valor ASCII ubicado en Datos_IoT convertirlo a binario y poner el resultado en un arreglo DATOS_BIN. La subrutina recibe del programa principal las direcciones Datos_IoT y Datos_BIN por medio de la pila y recibe por memoria el valor de la cantidad de datos a procesar (CANT). Para ello el programa principal debe apilar DATOS_BIN y Datos_IoT, en ese orden, antes de llamar la subrutina, debe usar Load Effective Address administrar los punteros en la pila. Además de las estructuras de datos indicadas, esta subrutina únicamente utilizará CONT, OFFSET y ACC. Para barrer los arreglos se debe utilizar direccionamiento indexado por acumulador.

Estructuras de datos:
cant: variable tipo byte donde se guarda la cantidad de datos a procesar
cont: variable tipo byte, contiene los datos procesados
offset: variable tipo byte, recorre la tabla
acc: variable tipo word, almacena el resultado



Subrutina Mover: Esta subrutina es la encargada de separar los patrones de 12 bits en sus 3 "nibbles" constitutivos, colocando cada parte en los arreglos definidos para tal fin. Los "nibbles" deben ser colocados en el mismo orden en que aparecen en Datos_BIN. La subrutina recibe la dirección Datos_BIN por medio del registro índice X y las direcciones de Nibble_UP, Nibble_MED y Nibble_LOW por medio de tres punteros (variables) ubicadas en las direcciones con el mismo nombre de los arreglos. Además recibe por memoria el valor de la cantidad de datos a procesar (CANT).

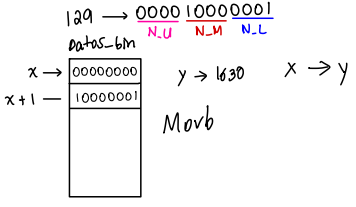
① Se debe de tomar los datos que están en Datos_BIN

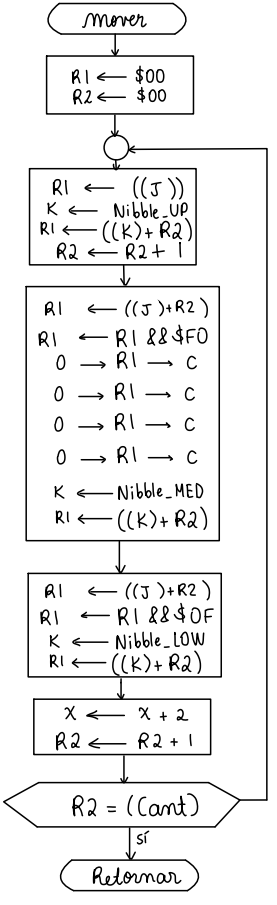
② Saber la cantidad de datos que tengo en cant

③ Separar cada dato de 12 bits en 4 bits, los primeros 4 (Nibble_LOW), los del medio (Nibble_MED) y los últimos (Nibble_UP)

④ Guardar cada uno de los cuatro bits en su respectiva variable de dirección.

⑤ Actualiza para saber si ya proceso todos.





Estructuras de datos:

Cant: variable tipo byte donde se guarda la cantidad de datos a procesar

Nibble_UP: constante tipo word, puntero apunta dirección Nibble_UP

Nibble_MED: constante tipo word, puntero apunta dirección Nibble_MED

Nibble_LOW: constante tipo word, puntero apunta dirección Nibble_LOW

Subrutina IMPRIMIR: Esta subrutina es llamada desde el programa principal y debe imprimir los resultados de la siguiente manera:

>CANTIDAD DE VALORES PROCESADOS: <CONT>

Este resultado debe aparecer dos líneas por debajo del último mensaje desplegado.

Adicionalmente esta subrutina deberá imprimir, en el terminal, los arreglos Nibble_UP, Nibble_MED y Nibble_LOW calculados, de la siguiente manera.

> Nibble_UP: <nibbleUp_1> ,<nibbleUp_2>, …, <nibbleUp_k>

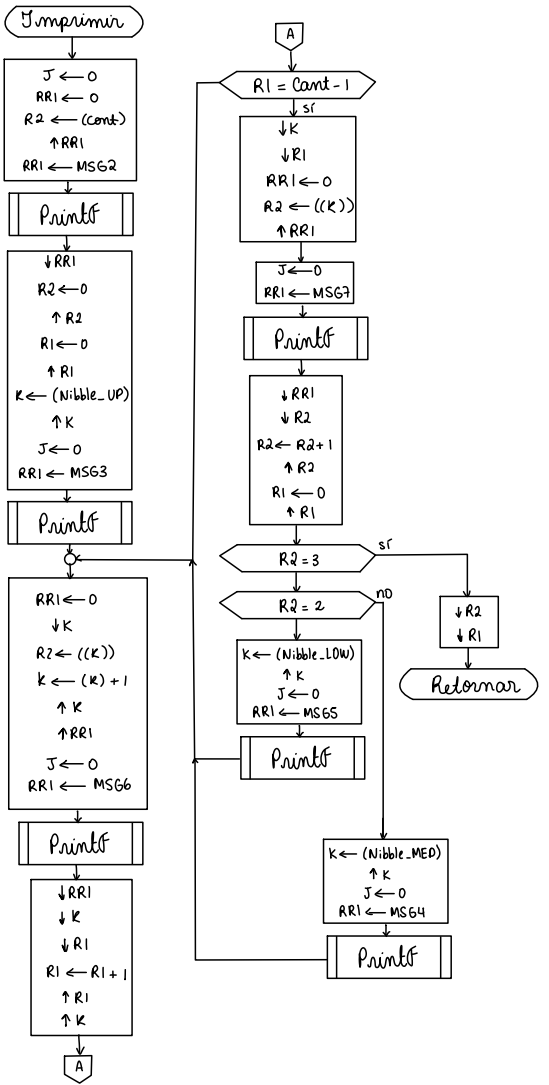
> Nibble_MED: <nibbleMed_1> ,<nibbleMed_2>, …, <nibbleMed_k>

> Nibble_LOW: <nibbleLow_1> ,<nibbleLow_2>, …, <nibbleLow_k>

Note que los datos de los arreglos deben imprimirse separados por una coma, excepto el último. Estos resultados deberán aparecer dos líneas por debajo del último mensaje desplegado y separados dos líneas entre sí. Los datos de los arreglos deben imprimirse en hexadecimal. En estas subrutinas solo pueden usarse las estructuras de datos indicadas además de CANT.

MSG2 : Cantidad de valores procesados:
MSG3 : Nibble_UP:
MSG4 : Nibble_MED:
MSG5 : Nibble_LOW:
MSG6 : 0%X,
MSG7 : 0%X

Estructuras de datos:
Nibble_UP: constante tipo word, puntero apunta dirección Nibble_UP
Nibble_MED: constante tipo word, puntero apunta dirección Nibble_MED
Nibble_LOW: constante tipo word, puntero apunta dirección Nibble_LOW
cont : variable tipo byte, contiene los datos procesados
MSG : cadena de caracteres



Entonces:

a. Diseñe el programa principal que haga el llamado a las subrutinas de tal manera que se resuelva el problema propuesto. Diseñe las subrutinas según lo indicado. Depure su solución al máximo procurando realizar el diseño más pequeño posible. Se evaluará eficiencia y eficacia de la solución. **Solo debe utilizar las estructuras de datos indicadas en el enunciado.**
b. Codifique el diseño realizado, depure el código hasta llevarlo a la versión más pequeña posible. Utilice la Tabla de Estructuras de datos indicada. Los punteros Nibble_UP, Nibble_MED y Nibble_LOW debe ser inicializados en tiempo de ejecución con las direcciones $1630, $1660 y $1690. El código del programa se debe colocar a partir de la dirección $2000. El código debe crear la tabla Datos_IoT con los siguientes datos:

Dato1: "129"	Dato6: "1536"	Dato11: "0"	Dato16: "64"
Dato2: “729"	Dato7: “534"	Dato12: "1329"	Dato17: "128"
Dato3: “3954"	Dato8: “2755"	Dato13: "1783"	Dato18: "256"
Dato4: “1875"	Dato9: “2021"	Dato14: "9"	Dato19: "512"
Dato5: “75"	Dato10: "389"	Dato15: "2804"	Dato20: "4095"

c. Compruebe la correcta operación de su programa utilizando la tarjeta Dragon 12.

Remita el **código fuente únicamente** (Archivo .asm) de su programa, así como el documento pdf con los diseños, explicaciones, notas de cálculo, etc. El formato del nombre del archivo fuente debe ser SUNOMBRE_T3.asm y el documento pdf debe nombrarse SUNOMBRE_T3.pdf. Se revisará eficiencia y eficacia del diseño y su codificacion, así como la correcta operación según las especificaciones indicadas, aplicando un protocolo de pruebas al diseño implementado.

Tabla de Estructuras de Datos

Estructura de datos	Descripción	Dirección
Datos_IoT	Tabla con los datos fuente a ser procesados	$1500
Datos_BIN	Arreglo con el resultado de convertir los datos fuente a binario	$1600
CANT	Variable que contiene la cantidad de datos a procesar	$1000
CONT	Variable que contiene la cantidad de datos procesados	$1001
Offset	Variable auxiliar para almacenar el puntero de barrido de la tabla	$1002
ACC	Variable para almacenar el acumulado de cálculo en la conversión ASCII_BIN	$1003
Nibble_UP	Puntero que contiene la dirección del arreglo Nibble_Up	$1010
Nibble_MED	Puntero que contiene la dirección del arreglo Nibble_MED	$1012
Nibble_LOW	Puntero que contiene la dirección del arreglo Nibble_LOW	$1014
MSG	Mensajes ASCII	$1030