

Sistemas Embebidos

Practica 4. Lenguaje ensamblador del Atmega2560

Objetivos:

- Que el estudiante conozca el conjunto de instrucciones que soporta el MCU Atmega2560.
- Que el estudiante conozca las directivas básicas del ensamblador para el AVR Atmega2560
- Que el estudiante programe rutinas en lenguaje ensamblador para controlar un motor de pasos.

Material

- Software Atmel Studio 7
- Tarjeta protoboard
- Tarjeta Arduino Atmega
- Módulo Puente-H L298N
- 2 Resistencias de 10KΩ 1/2W
- 2 LEDs
- 2 Interruptores push-button
- Cables de conexión

1. Instrucciones del MCU Atmega2560

La Unidad de Procesamiento Central (CPU) del Microcontrolador (MCU) Atmega2560 tiene una arquitectura RISC (*Reduced Instruction Set Computing, Reducido Conjunto de Instrucciones*), la filosofía de los procesadores RISC es que tengan hardware menos complejo para soportar un conjunto de instrucciones reducido, esto permite que las instrucciones sean ejecutadas de una manera más eficiente y rápida, por lo regular una instrucción por ciclo del reloj. Los procesadores CISC (*Complex Instruction Set Computing*) son procesadores con arquitectura más compleja, tienen instrucciones de mayor nivel de abstracción, con el fin de que el compilador pueda llevar una instrucción de alto nivel a una instrucción a nivel de CPU y mejorar en lo general el desempeño de la computadora (pero con un hardware más complejo), es por esto que las arquitecturas CISC tienen un conjunto de instrucciones mucho más alto que una arquitectura RISC. Por ejemplo, una simple multiplicación, en una arquitectura RISC tendría que codificarse con cuatro instrucciones de un solo ciclo (dos lecturas de memoria a registros, una multiplicación y un almacenamiento de un registro a memoria), en cambio en una arquitectura CISC, lo realizaría con una sola instrucción de varios ciclos (instrucción de multiplicación con operandos en modo de direccionamiento indirecto y directo).

En las tablas 1, 2, 3 y 4, se muestra la mayoría de las instrucciones que soporta el MCU Atmega2560, agrupadas en instrucciones aritméticas y lógicas, instrucciones de transferencia de datos, instrucciones de control e instrucciones a nivel de bit.

Tabla 1.1. Instrucciones Aritméticas y Lógicas

| Mnemónico | Pseudocódigo | Nombre | Descripción |
|------------------|------------------------------------|--------------------------------------|---|
| ADD Rd,Rr | $Rd \leftarrow Rd + Rr$ | Suma sin acarreo | Suma dos registros sin el bit de acarreo $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| ADC Rd,Rr | $Rd \leftarrow Rd + Rr + C$ | Suma con acarreo | Suma dos registros y el bit de acarreo $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| ADIW Rd,k | $Rd+1: Rd \leftarrow Rd+1: Rd + k$ | Suma de 16-bits con constante | Suma una constante con una palabra almacenada en un par de registros $Rd \in \{R24, R26, R28, R30\}$ $k \in \{0, 1, 2, \dots, 63\}$ |
| SUB Rd,Rr | $Rd \leftarrow Rd - Rr$ | Resta sin acarreo | Resta el contenido de dos registros $Rd, Rr \in \{R0, R1, \dots, R31\}$ |

| | | | |
|---------------------|--------------------------------------|---|--|
| SUBI Rd,k | $Rd \leftarrow Rd - k$ | Resta de constante | Resta el contenido de un registro con una constante de 8 bits $Rd \in \{R16, R17, \dots, R31\}$ $k \in \{0, 1, \dots, 255\}$ |
| SBC Rd,Rr | $Rd \leftarrow Rd - Rr - C$ | Resta con acarreo | Resta dos registros y el bit de acarreo $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| SBCI Rd,k | $Rd \leftarrow Rd - k - C$ | Resta de constante con acarreo | Resta una constante a un registro y resta el bit de acarreo $Rd \in \{R16, R17, \dots, R31\}$ $k \in \{0, 1, 2, \dots, 255\}$ |
| SBIW Rd,k | $Rd+1: Rd \leftarrow Rd+1: Rd - k$ | Resta de 16-bits con acarreo | Resta una constante con una palabra almacenada en un par de registros $Rd \in \{R24, R26, R28, R30\}$ $k \in \{0, 1, 2, \dots, 63\}$ |
| AND Rd,Rr | $Rd \leftarrow Rd \wedge Rr$ | Y-lógica | Realiza una operación Y-Lógica a nivel de bit entre ambos registros $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| ANDI Rd,k | $Rd \leftarrow Rd \wedge k$ | Y-lógica con constante | Realiza una operación Y-lógica a nivel de bit con el contenido de un registro con una constante de 8 bits $Rd \in \{R16, R17, \dots, R31\}$ $k \in \{0, 1, \dots, 255\}$ |
| OR Rd,Rr | $Rd \leftarrow Rd \vee Rr$ | O-lógica | Realiza una operación O-Lógica a nivel de bit entre ambos registros $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| ORI Rd,k | $Rd \leftarrow Rd \vee k$ | O-lógica con constante | Realiza una operación O-lógica a nivel de bit con el contenido de un registro con una constante de 8 bits $Rd \in \{R16, R17, \dots, R31\}$ $k \in \{0, 1, \dots, 255\}$ |
| EOR Rd,Rr | $Rd \leftarrow Rd \oplus Rr$ | O Exclusiva-lógica | Realiza una operación O-Exclusiva- a nivel de bit entre ambos registros $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| COM Rd | $Rd \leftarrow \neg Rd$ | Complemento a 1 | Complemento a 1. Cambia ceros por unos y unos por ceros $Rd \in \{R0, R1, \dots, R31\}$ |
| NEG Rd | $Rd \leftarrow - Rd$ | Complemento a 2 | Complemento a 2. Cambia de signo el registro Rd $Rd \in \{R0, R1, \dots, R31\}$ |
| INC Rd | $Rd \leftarrow Rd + 1$ | Incremento | Incrementa a uno el registro Rd $Rd \in \{R0, R1, \dots, R31\}$ |
| DEC Rd | $Rd \leftarrow Rd - 1$ | Decremento | Decrementa en uno el registro Rd $Rd \in \{R0, R1, \dots, R31\}$ |
| CLR Rd | $Rd \leftarrow 0x00$ | Limpiar un registro | Establece a cero todos los bits del Rd $Rd \in \{R0, R1, \dots, R31\}$ |
| SER Rd | $Rd \leftarrow 0xFF$ | Enciende todos los bits de un registro | Establece a uno todos los bits del Rd $Rd \in \{R16, R17, \dots, R31\}$ |
| CBR Rd, k | $Rd \leftarrow Rd \wedge (\neg k)$ | Apagar bits en un registro | Establece a cero los bits especificados por k, en el registro Rd. $Rd \in \{R16, R17, \dots, R31\}$ $k \in \{0, 1, \dots, 255\}$ |
| SBR Rd, k | $Rd \leftarrow Rd \vee k$ | Enciende bits en un registro | Establece a uno los bits especificados por k, en el registro Rd. $Rd \in \{R16, R17, \dots, R31\}$ $k \in \{0, 1, \dots, 255\}$ |
| TST Rd | $Rd \leftarrow Rd \wedge Rd$ | Prueba de cero o negativo | Prueba si el registro Rd es cero o negativo. El registro Rd permanece sin cambio $Rd \in \{R0, R1, \dots, R31\}$ |
| MUL Rd, Rr | $R1:R0 \leftarrow Rd \times Rr$ (UU) | Multiplicación sin signo | Multiplicación de 8-bits sin signo entre Rd y Rr, el resultado se almacena en R1:R0 $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| MULS Rd, Rr | $R1:R0 \leftarrow Rd \times Rr$ (SS) | Multiplicación con signo | Multiplicación de 8-bits con signo entre Rd y Rr, el resultado se almacena en R1:R0 $Rd, Rr \in \{R16, R17, \dots, R31\}$ |
| MULSU Rd, Rr | $R1:R0 \leftarrow Rd \times Rr$ (SU) | Multiplicación con y sin signo | Multiplicación de 8-bits con signo y sin signo entre Rd y Rr, el resultado se almacena en R1:R0 con signo. $Rd, Rr \in \{R16, R17, \dots, R31\}$ |

| | | | | |
|----------------------|--|---------------------------------------|-----------|--|
| FMUL Rd, Rr | $R1:R0 \leftarrow (Rd \times Rr) \ll 1$ (UU) | Multiplicación sin signo | Q7 | Multiplicación fraccionaria Q7 de 8-bits sin signo entre Rd y Rr, el resultado se almacena en R1:R0 en Q15 $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| FMULS Rd, Rr | $R1:R0 \leftarrow (Rd \times Rr) \ll 1$ (SS) | Multiplicación con signo | Q7 | Multiplicación fraccionaria Q7 de 8-bits con signo entre Rd y Rr, el resultado se almacena en R1:R0 en Q15 $Rd, Rr \in \{R16, R17, \dots, R31\}$ |
| FMULSU Rd, Rr | $R1:R0 \leftarrow (Rd \times Rr) \ll 1$ (SU) | Multiplicación sin y sin signo | Q7 | Multiplicación fraccionaria Q7 de 8-bits con signo y sin signo entre Rd y Rr, el resultado se almacena en R1:R0 con signo en Q15. $Rd, Rr \in \{R16, R17, \dots, R31\}$ |

Tabla 1.2. Instrucciones de transferencias de datos

| Mnemónico | Pseudocódigo | Nombre | Descripción |
|---------------------------|---|---|--|
| MOV Rd, Rr | $Rd \leftarrow Rr$ | Copiar un registro | Copiar el contenido del registro Rr en Rd. $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| MOVW Rd, Rr | $Rd+1:Rd \leftarrow Rr+1:Rr$ | Copia registro de 16 bits. | Copia el contenido de un par de registros a otro par de registro en modo de 16-bits $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| LDI Rd, k | $Rd \leftarrow k$ | Carga inmediata | Carga al registro Rr con la constante k en modo de direccionamiento inmediato. $Rd \in \{R16, R17, \dots, R31\}$ $k \in \{0, 1, 2, \dots, 255\}$ |
| LDS Rd, k | $Rd \leftarrow *(k)$ | Carga de forma directa | Carga el registro Rd, con el contenido de la dirección de memoria de datos apuntada por k $Rd \in \{R0, R1, \dots, R31\}$ $k \in \{0, 1, 2, \dots, 65535\}$ |
| LD Rd, indexR | $Rd \leftarrow *(indexR)$ | Carga indirecta de la memoria de datos | Carga al registro Rd, con el contenido de la casilla de memoria apuntada por el registro índice indexR. $Rd \in \{R0, R1, \dots, R31\}$ $indexR \in \{X, Y, Z\}$ |
| LD Rd, indexR+ | $Rd \leftarrow *(indexR)$ $indexR \leftarrow indexR + 1$ | Carga indirecta de la memoria de datos con post-incremento | Carga al registro Rd, el contenido de la casilla de memoria de datos apuntada por el registro índice indexR y después se incrementa el registro indexR. $Rd \in \{R0, R1, \dots, R31\}$ $indexR \in \{X, Y, Z\}$ |
| LD Rd, -indexR | $indexR \leftarrow indexR - 1$ $Rd \leftarrow *(indexR)$ | Carga indirecta de la memoria de datos con pre-decremento | Decrementa en uno al registro indexR, después, se carga al registro Rr, con el contenido de la casilla de memoria de datos apuntada por el registro índice indexR. $Rd \in \{R0, R1, \dots, R31\}$ $indexR \in \{X, Y, Z\}$ |
| LDD Rd, indexR + d | $Rd \leftarrow *(indexR + d)$ | Carga indirecta de la memoria de datos con desplazamiento | Carga al registro Rr, con el contenido de la casilla de memoria que tiene la dirección obtenida con el resultado de la suma del registro índice indexR más la constante d, $Rd \in \{R0, R1, \dots, R31\}$ $indexR \in \{Y, Z\}$ $d \in \{0, 1, 2, \dots, 63\}$ |
| STS k, Rr | $*(k) \leftarrow Rr$ | Almacenamiento en forma directa a memoria de datos | Almacena el registro Rr, en la casilla de memoria de datos cuya dirección es k $Rd \in \{R0, R1, \dots, R31\}$ $k \in \{0, 1, 2, \dots, 65535\}$ |
| ST indexR, Rr | $*(indexR) \leftarrow Rr$ | Almacenamiento en forma indirecta a la memoria de datos | Almacena el registro Rr en la casilla de memoria de datos apuntada por el registro índice indexR. $Rd \in \{R0, R1, \dots, R31\}$ $indexR \in \{X, Y, Z\}$ |

| | | | |
|---------------------------|---|--|--|
| ST indexR+, Rr | $*(\text{indexR}) \leftarrow \text{Rr}$ $\text{indexR} \leftarrow \text{indexR} + 1$ | Almacenamiento en forma indirecta en la memoria de datos con post-incremento | Almacena el registro Rr en la casilla de memoria de datos apuntada por el registro índice indexR y después se incrementa el registro indexR. $\text{Rd} \in \{\text{R0}, \text{R1}, \dots, \text{R31}\}$ $\text{indexR} \in \{\text{X}, \text{Y}, \text{Z}\}$ |
| ST -indexR, Rr | $\text{indexR} \leftarrow \text{indexR} - 1$ $*(\text{indexR}) \leftarrow \text{Rr}$ | Almacenamiento en forma indirecta en la memoria de datos con pre-decremento | Decrementa al registro índice indexR en uno y después, almacena el registro Rr en la casilla de memoria de datos con dirección apuntada por el registro índice indexR. $\text{Rd} \in \{\text{R0}, \text{R1}, \dots, \text{R31}\}$ $\text{indexR} \in \{\text{X}, \text{Y}, \text{Z}\}$ |
| STD indexR + d, Rr | $*(\text{indexR} + d) \leftarrow \text{Rr}$ | Almacenamiento en forma indirecta en la memoria de datos con desplazamiento | Almacena el registro Rr en la casilla de memoria que tiene la dirección obtenida con el resultado de la suma del registro índice indexR más la constante d, $\text{Rd} \in \{\text{R0}, \text{R1}, \dots, \text{R31}\}$ $\text{indexR} \in \{\text{Y}, \text{Z}\}$ $d \in \{0, 1, 2, \dots, 63\}$ |
| LPM | $\text{R0} \leftarrow *(Z)$ | Carga indirecta de la memoria de programa | Carga al registro R0, con el contenido de la casilla de memoria de programa apuntada por el registro índice Z. |
| LPM Rd, Z | $\text{Rd} \leftarrow *(Z)$ | Carga indirecta de la memoria de programa | Carga al registro Rd, con el contenido de la casilla de memoria programa apuntada por el registro índice Z. $\text{Rd} \in \{\text{R0}, \text{R1}, \dots, \text{R31}\}$ |
| LD Rd, Z+ | $\text{Rd} \leftarrow *(Z)$ $Z \leftarrow Z + 1$ | Carga indirecta de la memoria de programa con post-incremento | Carga al registro Rd, con el contenido de la casilla de memoria de programa apuntada por el registro índice Z, después se incrementa el registro Z en uno. $\text{Rd} \in \{\text{R0}, \text{R1}, \dots, \text{R31}\}$ |
| SPM | $*(\text{RAMPZ:Z}) \leftarrow \text{R1:R0}$ | Almacenamiento en forma indirecta a la memoria de programa | Almacena el dato de 16-bits contenido en los registros R1:R0, en la casilla de memoria de programa con dirección apuntada por el registro índice Z. NOTA: Para escribir en la memoria flash, es necesario realizar un procedimiento específico |
| SPM Z, Rr | $*(Z) \leftarrow \text{Rr}$ | Almacenamiento en forma indirecta a la memoria de programa | Almacena el registro Rr en la casilla de memoria de programa apuntada por el registro índice Z. NOTA: Para escribir en la memoria flash, es necesario realizar un procedimiento específico $\text{Rd} \in \{\text{R0}, \text{R1}, \dots, \text{R31}\}$ |
| SPM Z+, Rr | $*(Z) \leftarrow \text{Rr}$ $Z \leftarrow Z + 1$ | Almacenamiento en forma indirecta en la memoria de programa con post-incremento | Almacena el registro Rr en la casilla de memoria de programa apuntada por el registro índice Z y después se incrementa el registro Z en uno. $\text{Rd} \in \{\text{R0}, \text{R1}, \dots, \text{R31}\}$ |
| IN Rd,k | $\text{Rd} \leftarrow *(k)$ $k \in \{\text{I/O Registers}\}$ | Carga un desde el espacio de direcciones I/O Reg | Carga el registro Rd, con el contenido de la casilla de memoria situada en la dirección relativa a la sección I/O dada por k. $\text{Rd} \in \{\text{R0}, \text{R1}, \dots, \text{R31}\}$ $k \in \{0, 1, 2, \dots, 63\}$ |
| OUT k, Rr | $*(k) \leftarrow \text{Rr}$ $k \in \{\text{I/O Registers}\}$ | Almacenamiento hacia el espacio de memoria I/O | Almacena el registro Rd, en la casilla de memoria determinada por la dirección relativa k en el espacio I/O $\text{Rd} \in \{\text{R0}, \text{R1}, \dots, \text{R31}\}$ $k \in \{0, 1, 2, \dots, 63\}$ |
| PUSH Rr | $*(\text{SP}) \leftarrow \text{Rr}$ $\text{SP} \leftarrow \text{SP} - 1$ | Guarda un registro a la pila | Guarda el registro Rr en la casilla de memoria apuntada por el registro SP (<i>Stack Pointer</i>), después se decrementa en uno el registro SP. |

| | | | |
|------------------|---|---------------------------|---|
| | | | $Rd \in \{R0, R1, \dots, R31\}$ |
| POP Rd | $SP \leftarrow SP + 1$ $Rd \leftarrow *(SP)$ | Extrae un dato de la pila | Incrementa en uno el registro SP (<i>Stack Pointer</i>), después extrae el dato del tope de la pila y lo almacena en el registro Rd. $Rd \in \{R0, R1, \dots, R31\}$ |
| XCH Z, Rd | $*(Z) \leftarrow Rd$ $Rd \leftarrow *(Z)$ | Intercambio | Intercambia el contenido del registro Rd con un dato en memoria de datos apuntado por el registro índice Z. $Rd \in \{R0, R1, \dots, R31\}$ |
| LAS Rd | $*(Z) \leftarrow Rd \vee *(Z)$ $Rd \leftarrow *(Z)$ | Carga y Establece | Carga el registro Rd con el contenido de la casilla de memoria de datos apuntada por Z y enciende los bits, especificados en el registro Rd, de la misma casilla de memoria. $Rd \in \{R0, R1, \dots, R31\}$ |
| LAC Rd | $*(Z) \leftarrow \neg Rd \wedge *(Z)$ $Rd \leftarrow *(Z)$ | Carga y Limpia | Carga el registro Rd con el contenido de la casilla de memoria de datos apuntada por Z y apaga los bits, especificados en el registro Rd, de la misma casilla de memoria. $Rd \in \{R0, R1, \dots, R31\}$ |
| LAT Rd | $*(Z) \leftarrow Rd \oplus *(Z)$ $Rd \leftarrow *(Z)$ | Carga y conmuta | Carga el registro Rd con el contenido de la casilla de memoria de datos apuntada por Z y conmuta los bits, especificados en el registro Rd, de la misma casilla de memoria. $Rd \in \{R0, R1, \dots, R31\}$ |
| | | | |

Tabla 1.3. Instrucciones de salto


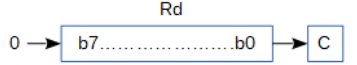
| Mnemónico | Pseudocódigo | Nombre | Descripción |
|----------------|---|--------------------------------------|---|
| RJMP k | $PC \leftarrow PC + k + 1$ | Salto relativo | Salto relativo, el contador de programa PC se incrementa a la dirección de memoria de programa dada por $PC + k + 1$ |
| IJMP | $PC(15:0) \leftarrow Z$ | Salto indirecto | Salto indirecto, el contador de programa PC se actualiza con la dirección de programa dada por el registro índice Z. |
| EIJMP | $PC(15:0) \leftarrow Z$ $PC(21:16) \leftarrow EIND$ | Salto indirecto extendido | Salto indirecto de forma extendida, el contador de programa PC se actualiza con la dirección de programa formada por la unión de los registros EIND:Z. |
| JMP k | $PC \leftarrow k$ | Salto incondicional | Salto incondicional, el contador de programa PC se actualiza con el valor de la constante k |
| RCALL k | $*(SP) \leftarrow PC + 1$ $SP \leftarrow SP - 1$ $PC \leftarrow PC + k + 1$ | Llamada relativa a subrutina | Llamada relativa a subrutina, La dirección de regreso (La dirección de la siguiente instrucción de RCALL) es almacenada en la pila y se actualiza el PC a la dirección donde está definida la subrutina ($PC + k + 1$). |
| ICALL k | $*(SP) \leftarrow PC + 1$ $SP \leftarrow SP - 1$ $PC(15:0) \leftarrow Z$ | Llamada indirecta a subrutina | Llamada indirecta a subrutina, La dirección de regreso (La dirección de la siguiente instrucción de ICALL) es almacenada en la pila y se actualiza el PC a la dirección almacenada en el registro índice Z. |
| CALL k | $*(SP) \leftarrow PC + 1$ $SP \leftarrow SP - 1$ $PC \leftarrow k$ | Llamada a subrutina | Llamada a subrutina, La dirección de regreso (La dirección de la siguiente instrucción de CALL) es almacenada en la pila y se actualiza el PC con la constante k. |
| RET | $PC \leftarrow *(SP)$ $SP \leftarrow SP + 1$ | Regreso de subrutina | Regreso de subrutina, se extrae de la pila la dirección de regreso de subrutina y es almacenada en el PC |

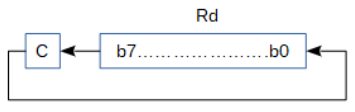
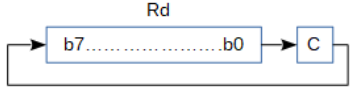
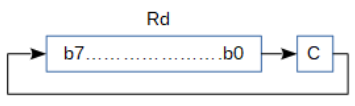
| | | | |
|--------------------|---|--|--|
| RETI | $PC \leftarrow *(SP)$ $SP \leftarrow SP + 1$ $SREG.I \leftarrow 1$ | Regreso de subrutina | Regreso de interrupción, se extrae de la pila la dirección de regreso de interrupción y es almacenada en el PC. Se activa la bandera de interrupción I en el registro SREG. |
| CP Rd, Rr | $Rd - Rr$ | Comparación | Compara dos registros. Esta instrucción realiza la operación de resta, sin realizar ninguna modificación en los registros, su función es solo para activar las banderas en el registro de estado (SREG) $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| CPC Rd, Rr | $Rd - Rr - C$ | Comparación con acarreo | Compara dos registros con acarreo. Esta instrucción realiza la operación de resta con acarreo, sin realizar ninguna modificación en los registros, su función es solo para activar las banderas en el registro de estado (SREG) $Rd, Rr \in \{R16, R17, \dots, R31\}$ |
| CPI Rd, k | $Rd - k$ | Comparación con una constante | Compara el registro Rd con la constante k. Esta instrucción realiza la operación de resta, sin alterar al registro Rd, su función es solo para activar las banderas en el registro de estado (SREG) $k \in \{0, 1, \dots, 255\}$ $Rd \in \{R0, R1, \dots, R31\}$ |
| CPSE Rd, Rr | If (Rd - Rr) $PC \leftarrow PC + 2$ else $PC \leftarrow PC + 1$ | Comparación y saltar la próxima instrucción | Compara dos registros, si son iguales entonces salta a la segunda instrucción que le sigue, de lo contrario continua con la instrucción siguiente. Su función principal, es junto con una instrucción de salto, realizar fácilmente estructuras de control . $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| SBRC Rr, b | If (Rr.b = 0) $PC \leftarrow PC + 2$ else $PC \leftarrow PC + 1$ | Saltar la próxima instrucción si un bit es cero. | Salta la próxima instrucción si el bit b en el registro Rr es cero, de lo contrario continua con la instrucción siguiente. Esta instrucción, junto con una instrucción de salto, permite realizar fácilmente estructuras de control . $Rr \in \{R0, R1, \dots, R31\}$ $b \in \{0, 1, \dots, 7\}$ |
| SBRS Rr, b | If (Rr.b = 1) $PC \leftarrow PC + 2$ else $PC \leftarrow PC + 1$ | Saltar la próxima instrucción si un bit es uno. | Salta la próxima instrucción si el bit b en el registro Rr es uno, de lo contrario continua con la instrucción siguiente. Esta instrucción, junto con una instrucción de salto, permite realizar fácilmente estructuras de control . $Rr \in \{R0, R1, \dots, R31\}$ $b \in \{0, 1, \dots, 7\}$ |
| SBIC k, b | If (*(k).b = 0) $PC \leftarrow PC + 2$ else $PC \leftarrow PC + 1$ | Saltar la próxima instrucción si un bit de un registro I/O es cero. | Salta la próxima instrucción si el bit b en el registro I/O con dirección relativa k es cero, de lo contrario continua con la instrucción siguiente. Esta instrucción, junto con una instrucción de salto, permite realizar fácilmente estructuras de control . $k \in \{0, 1, \dots, 31\}$ $b \in \{0, 1, \dots, 7\}$ |
| SBIS k, b | If (*(k).b = 1) | Saltar la próxima | Salta la próxima instrucción si el bit b en el |

| | | | |
|--------------------|---|---|--|
| | $PC \leftarrow PC + 2$ else $PC \leftarrow PC + 1$ | instrucción si un bit de un registro I/O es uno. | registro I/O con dirección relativa k es uno, de lo contrario continua con la instrucción siguiente. Esta instrucción, junto con una instrucción de salto, permite realizar fácilmente estructuras de control . $k \in \{0, 1, \dots, 31\}$ $b \in \{0, 1, \dots, 7\}$ |
| BRBS b, k | If (SREG. $b = 1$) $PC \leftarrow PC + k + 1$ else $PC \leftarrow PC + 1$ | Saltar si un bit de bandera está activado. | Salto relativo condicional, salta a una dirección relativa si el bit de bandera b en el registro de estado SREG esta activado. $k \in \{-64, -63, \dots, 62, 63\}$ $b \in \{0, 1, \dots, 7\}$ |
| BRBC b, k | If (SREG. $b = 0$) $PC \leftarrow PC + k + 1$ else $PC \leftarrow PC + 1$ | Saltar si un bit de bandera está en cero. | Salto relativo condicional, salta a una dirección relativa si el bit de bandera b en el registro de estado SREG es cero. $k \in \{-64, -63, \dots, 62, 63\}$ $b \in \{0, 1, \dots, 7\}$ |
| BREQ k | If ($Z = 1$) $PC \leftarrow PC + k + 1$ else $PC \leftarrow PC + 1$ | Saltar si son iguales | Salto si son iguales, Si el bit bandera Z en el registro SREG es 1, salta a la posición relativa $PC + k + 1$, de lo contrario continua con la siguiente instrucción. Por lo regular, esta instrucción es ejecutada después de una instrucción de comparación como CP, CPI, etc. $k \in \{-64, -63, \dots, 62, 63\}$ |
| BRNE k | If ($Z = 0$) $PC \leftarrow PC + k + 1$ else $PC \leftarrow PC + 1$ | Saltar si no son iguales | Salto si no son iguales, Si el bit bandera Z en el registro SREG es 0, salta a la posición relativa $PC + k + 1$, de lo contrario continua con la siguiente instrucción. Por lo regular, esta instrucción es ejecutada después de una instrucción de comparación como CP, CPI, etc. $k \in \{-64, -63, \dots, 62, 63\}$ |
| BRCS k | If ($C = 1$) $PC \leftarrow PC + k + 1$ else $PC \leftarrow PC + 1$ | Saltar si la bandera de acarreo está activa | Salto condicional relativo, Si el bit bandera C en el registro SREG es 1, salta a la posición relativa $PC + k + 1$, de lo contrario continua con la siguiente instrucción. $k \in \{-64, -63, \dots, 62, 63\}$ |
| BRCC k | If ($C = 0$) $PC \leftarrow PC + k + 1$ else $PC \leftarrow PC + 1$ | Saltar si la bandera de acarreo es cero | Salto condicional relativo, Si el bit bandera C en el registro SREG es 0, salta a la posición relativa $PC + k + 1$, de lo contrario continua con la siguiente instrucción. $k \in \{-64, -63, \dots, 62, 63\}$ |
| BRLO k | If ($C = 1$) $PC \leftarrow PC + k + 1$ else $PC \leftarrow PC + 1$ | Saltar si es menor (sin signo) | Salto condicional relativo, Si el bit bandera C en el registro SREG es 1, salta a la posición relativa $PC + k + 1$, de lo contrario continua con la siguiente instrucción. Por lo regular, esta instrucción es ejecutada después de una instrucción de comparación como CP, CPI, etc., para comparar si un número entero sin signo es menor que otro. $k \in \{-64, -63, \dots, 62, 63\}$ |
| BRMI k | If ($N = 1$) $PC \leftarrow PC + k + 1$ else $PC \leftarrow PC + 1$ | Saltar si es negativo | Salto condicional relativo, Si el bit bandera N en el registro SREG es 1, salta a la posición relativa $PC + k + 1$, de lo contrario continua con la siguiente instrucción. Por lo regular, esta instrucción es ejecutada después de una |

| | | | |
|---------------|--|-----------------------------------|---|
| | | | instrucción de comparación como CP, CPI, etc., para comparar si un número entero con signo es menor que otro. $k \in \{-64, -63, \dots, 62, 63\}$ |
| BRPL k | If (N = 0) PC \leftarrow PC + k + 1 else PC \leftarrow PC + 1 | Saltar si es positivo | Salto condicional relativo, Si el bit bandera N en el registro SREG es 0, salta a la posición relativa PC + k + 1, de lo contrario continua con la siguiente instrucción. Por lo regular, esta instrucción es ejecutada después de una instrucción de comparación como CP, CPI, etc., para comparar si un número entero con signo es mayor que otro. $k \in \{-64, -63, \dots, 62, 63\}$ |
| BRSB k | If (C = 0) PC \leftarrow PC + k + 1 else PC \leftarrow PC + 1 | Saltar si es mayor o igual | Salto condicional relativo, Si el bit bandera C en el registro SREG es 0, salta a la posición relativa PC + k + 1, de lo contrario continua con la siguiente instrucción. Por lo regular, esta instrucción es ejecutada después de una instrucción de comparación como CP, CPI, etc., para comparar si un número entero sin signo es mayor o igual que otro. $k \in \{-64, -63, \dots, 62, 63\}$ |
| BRGE k | If (S = 0) PC \leftarrow PC + k + 1 else PC \leftarrow PC + 1 | Saltar si es mayor o igual | Salto condicional relativo, Si el bit bandera S en el registro SREG es 0, salta a la posición relativa PC + k + 1, de lo contrario continua con la siguiente instrucción. Por lo regular, esta instrucción es ejecutada después de una instrucción de comparación como CP, CPI, etc., para comparar si un número entero con signo es mayor o igual que otro. $k \in \{-64, -63, \dots, 62, 63\}$ |
| BRLT k | If (S = 1) PC \leftarrow PC + k + 1 else PC \leftarrow PC + 1 | Saltar si es mayor o igual | Salto condicional relativo, Si el bit bandera S en el registro SREG es 1, salta a la posición relativa PC + k + 1, de lo contrario continua con la siguiente instrucción. Por lo regular, esta instrucción es ejecutada después de una instrucción de comparación como CP, CPI, etc., para comparar si un número entero con signo es menor que otro. $k \in \{-64, -63, \dots, 62, 63\}$ |
| | | | |

Tabla 1.4. Instrucciones a nivel de bit

| Mnemónico | Pseudocódigo | Nombre | Descripción |
|---------------|--|--|--|
| LSL Rd | C \leftarrow Rd(7) Rd \leftarrow Rd << 1  | Corrimiento lógico hacia la izquierda | Hacer un corrimiento a la izquierda de los bits del registro Rd, el bit 7 se amacena en el bit de acarreo (C) Rd, Rr $\in \{R0, R1, \dots, R31\}$ |
| LSR Rd | C \leftarrow Rd(0) Rd \leftarrow Rd >> 1  | Corrimiento lógico hacia la derecha | Hacer un corrimiento a la derecha de los bits del registro Rd, el bit 0 se amacena en el bit de acarreo (C) Rd, Rr $\in \{R0, R1, \dots, R31\}$ |

| | | | |
|--|--|---|--|
| ROL Rd | $C \leftarrow Rd(7)$ $Rd \leftarrow Rd \ll 1$  | Corrimiento rotatorio de bits hacia la izquierda | Hacer un corrimiento a la izquierda de los bits del registro Rd, el bit 7 se amacena en el bit de acarreo (C) y este a su vez es almacenado en el bit 0 $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| ROR Rd | $C \leftarrow Rd(0)$ $Rd \leftarrow Rd \gg 1$  | Corrimiento rotatorio hacia la derecha | Hacer un corrimiento a la derecha de los bits del registro Rd, el bit 0 se amacena en el bit de acarreo (C) y este a su vez es almacenado en el bit 7 $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| ASR Rd | $Rd \leftarrow Rd \gg 1$ $Rd(7) \leftarrow Rd(6)$  | Corrimiento aritmético hacia la derecha | Hacer un corrimiento a la derecha de los bits del registro Rd, el bit 6 se amacena en el bit 7 (bit de signo) $Rd, Rr \in \{R0, R1, \dots, R31\}$ |
| SBI k,b | $*(k).b \leftarrow 1$ $k \in \{I/O \text{ Registers}\}$ | Enciende un bit en el registro I/O | Enciende el bit establecido por b en el registro I/O con dirección relativa k. $k \in \{0, 1, \dots, 63\}$ $b \in \{0, 1, \dots, 7\}$ |
| CBI k,b | $*(k).b \leftarrow 0$ $k \in \{I/O \text{ Registers}\}$ | Apaga un bit en el registro I/O | Apaga el bit establecido por b en el registro I/O con dirección relativa k. $k \in \{0, 1, \dots, 63\}$ $b \in \{0, 1, \dots, 7\}$ |
| BST Rr,b | $T \leftarrow Rr.b$ | Almacena un bit de un registro al bit T | Almacena un bit del registro Rr en el bit T del registro de estado SREG. $Rr \in \{R0, R1, \dots, R31\}$ $b \in \{0, 1, \dots, 7\}$ |
| BLD Rr,b | $Rr.b \leftarrow T$ | Carga un bit de un registro con el bit T | Carga un bit del registro Rr con el bit T del registro de estado SREG. $Rr \in \{R0, R1, \dots, R31\}$ $b \in \{0, 1, \dots, 7\}$ |
| BSET b | $SREG.b \leftarrow 1$ | Enciende un bit de bandera | Enciende el bit de bandera b del registro SREG. $b \in \{0, 1, \dots, 7\}$ |
| BCLR b | $SREG.b \leftarrow 0$ | Apaga un bit de bandera | Apaga el bit de bandera b del registro SREG. $b \in \{0, 1, \dots, 7\}$ |
| SEy $y \in \{C, N, Z, I, S, V, T, H\}$ | $SREG.y \leftarrow 1$ $y \in \{C, N, Z, I, S, V, T, H\}$ | Enciende el bit de bandera y | Enciende el bit de bandera y del registro SREG. $y \in \{C, N, Z, I, S, V, T, H\}$ |
| SLy $y \in \{C, N, Z, I, S, V, T, H\}$ | $SREG.y \leftarrow 0$ $y \in \{C, N, Z, I, S, V, T, H\}$ | Apaga el bit de bandera y | Apaga el bit de bandera y del registro SREG. $y \in \{C, N, Z, I, S, V, T, H\}$ |

2. Escritura en lenguaje ensamblador

Los programas en lenguaje ensamblador consisten en líneas de instrucciones, llamadas sentencias, cada línea puede representar: una instrucción del CPU, una directiva, una macro o simplemente un comentario. Los programas fuente para el ensamblador del AVR, están divididos en cuatro columnas que definen el lugar donde se debe colocar los elementos como: **Etiquetas**, **mnemónicos**, **directivas**, **lista de operandos** y **comentarios**, cada columna debe estar separada por lo menos por un espacio, las buenas prácticas de programación sugieren que se utilice tabuladores como separador. En particular, las líneas de código deben estar limitadas a 120 caracteres. En la siguiente figura se muestra los campos que forman una línea en para el ensamblador del AVR.

| [Etiqueta:] | Mnemónico/ Directiva | [lista de operando] | [;Comentarios] |
|-------------|-------------------------|---------------------|----------------|
|-------------|-------------------------|---------------------|----------------|

A continuación, se muestra un ejemplo de instrucciones en ensamblador

| | | |
|-----------|----------------------------------|--|
| | <code>.EQU VAR1=2</code> | <code>;Directiva que establece a la etiqueta VAR1 el valor 2</code> |
| | <code>.DSEG</code> | <code>;Establece el inicio de la sección de memoria de datos</code> |
| VAR2: | <code>.BYTE 1</code> | <code>;Reserva un byte de memoria de datos, su dirección es representada por la etiqueta VAR2</code> |
| | <code>.CSEG</code> | <code>;Inicia la sección de memoria de programa</code> |
| Etiqueta: | <code>LDI R30, LOW(VAR2)</code> | <code>;Carga la parte baja de la dirección representada por VAR2 al registro R30</code> |
| | <code>LDI R31, HIGH(VAR2)</code> | <code>;Carga la parte alta de la dirección representada por VAR2 al registro R31</code> |
| | <code>LD R1, Z</code> | <code>;Carga el contenido de la casilla VAR2 al registro R1</code> |
| | <code>ADD R1, R2</code> | <code>;Suma el contenido del registro R1 y el registro R2, el resultado es almacenado en R1</code> |

El primer campo **[Etiqueta]** es opcional, se incluye en el programa ensamblador, para referenciar a un valor constante o dirección de memoria. Ejemplo, la etiqueta VAR2, que representa la dirección de memoria de datos del byte reservado.

El campo **Mnemónico/Directiva** hace referencia al nombre de una instrucción que soporta el CPU o una **directiva** del ensamblador, por ejemplo, si se desea ejecutar la instrucción suma de dos números enteros, en el campo **mnemónico** se debe escribirse la instrucción **ADD**; por otro lado, si se desea reservar un byte en la memoria de datos, se debe utilizar la directiva **.BYTE**, esta directiva, le indica al programa ensamblador (no es una instrucción del CPU) que reserve un byte en la memoria de datos y su dirección se debe representar por una etiqueta.

El campo **[Lista de operandos]**, es opcional, contienen los operandos separados por comas de las instrucciones o directivas, . Por ejemplo, la instrucción **ADD** tienen como operandos los registros R1 y R0; por lo tanto, la instrucción completa quedará **ADDI R1, R2**; la instrucción suma el contenido del registro R1 con R2 y el resultado lo almacena en el registro R1.

El campo **[;comentarios]**, es opcional y solo se incluye para que el programador incluya alguna descripción acerca del código. Este campo debe ser precedido por punto y coma. Los comentarios son omitidos por el ensamblador.

Tabla 2.1. Descripción de algunas directivas soportadas por el programa ensamblador para AVR

| Directiva | Descripción |
|----------------------------------|---|
| .DSEG | Define el inicio de la sección de memoria de datos |
| .CSEG | Define el inicio de la sección de memoria de programa. |
| .ESEG | Define el inicio de la sección de memoria EEPROM. |
| .ORG expresión | Esta directiva establece la localización absoluta en la memoria de datos o memoria de programa donde iniciará el almacenamiento de datos o instrucciones. Por lo regular se emplea para definir la dirección de memoria donde especifica de inicio del programa. |
| .DEF Símbolo=Registro | La directiva DEF, permite referenciar a un registro con el nombre especificado en Símbolo. |
| .EQU Etiqueta=expresión | La directiva EQU, asigna un valor a una etiqueta, que puede ser usada posteriormente en el código. La etiqueta representa un valor constante y no puede ser cambiado. |
| .BYTE n | Reserva n bytes de memoria SRAM. La directiva BYTE debe ser precedida por una etiqueta. |
| .DB valor1, [...,valor_n] | Reserva n bytes de memoria de programa (FLASH o EEPROM) y los inicializa con los valores <i>valor1</i> , <i>valor2</i> , ..., <i>valor_n</i> . Esta directiva debe ser precedida por una etiqueta. |
| .DW valor1, [...,valor_n] | Reserva n palabras de 16-bits en la memoria de programa (FLASH o EEPROM) y los inicializa con los valores <i>valor1</i> , <i>valor2</i> , ..., <i>valor_n</i> . Esta directiva debe ser precedida por una etiqueta. |
| .INCLUDE "archivo" | Directiva que indica al ensamblador que se incluya el archivo especificado. |
| .MACRO | Directiva que indica el inicio de la definición de una macro. Una macro es la definición de código genérico para generar un código particular. Las macros pueden recibir hasta 10 parámetros de entrada @0-@9. Con la directiva .ENDMACRO se establece el final de la macro |

Para hacer más legible y fácil la escritura, el ensamblador incorpora expresiones en sus líneas de código. Las expresiones son un conjunto de operandos, operadores y funciones, que el propio ensamblador evalúa y el resultado lo sustituye por la expresión en las líneas de código.

Los operandos pueden ser representado en diferentes bases numéricas, por ejemplo, la contante entera 130 (decimal), se puede representar en hexadecimal por 0x82 o en binario por 0b10000010.

Algunas de las funciones que pueden ser utilizadas en el código son:

- LOW(expresión) regresa el byte menos significativo de la expresión.
- HIGH(expresión) regresa el segundo byte de la expresión.
- EXP2 (expresión) regresa $2^{\text{expresión}}$
- LOG2 (expresión) regresa la parte entera de $\log_2(\text{expresión})$

En las expresiones, también se pueden incluir operadores como: *, /, +, -, <<, >>, <, <=, >, >=, ==, !=, , &&, || !, &, |, ~, ^.

3. Realización de sentencias de control

Con el conjunto de instrucciones que soporta el Atmega2560, se puede implementar las diferentes sentencias de control que se encuentran en la mayoría de los lenguajes de programación de alto nivel como son: sentencias IF, FOR, WHILE y DO-WHILE.

3.1. Realización en ensamblador de la estructura de decisión.

La estructura de decisión permite ejecutar un bloque de instrucciones, de entre varios bloques si se cumple una determinada condición. Existen tres tipos de estructuras de decisión: *simple (if-then)*, *doble (if-else)* y *múltiple (if-else-if, switch)*. En las siguientes tablas se muestra la forma de codificar en lenguaje ensamblador, las estructuras de decisión *if-then* y *if-else*.

Tabla 3.1. Codificación de ensamblador de la estructura de control IF-THEN

| Estructura IF-THEN | Realización en ensamblador |
|---|---|
| <pre> graph TD Entry(()) --> Cond{Si Cond} Cond -- Verdadero --> Acciones[Acciones Verdaderas] Cond -- Falso --> Merge(()) Acciones --> Merge Merge --> Exit(()) </pre> | <p>; 1ª Forma: ; Por ejemplo, si la condición es R2 == R3</p> <p>CPSE R2, R3 ;Si R2 == R3 entonces la siguiente ;instrucción se la salta RJMP END_IF ;salto relativo</p> <p>ADD R0, R3 ;Acciones verdaderas SUB R0, R5 INC R10</p> <p>END_IF: ;Fin del if</p> |
| | <p>; 2ª forma ; Con el uso de la instrucción CP ; e instrucciones de salto condicional</p> <p>; Por ejemplo, si la condición es R2 == R3 CP R2,R3 ;comparación de R2 y R3 BRNE FIN_SI ;Si R2 != R3 salta FIN_SI</p> <p>ADD R0, R3 ;Acciones verdaderas SUB R0, R5 INC R10</p> <p>FIN_SI: ;Fin de la estructura IF</p> |

Tabla 3.2. Codificación de ensamblador de la estructura de control IF-ELSE

| Estructura IF-ELSE | Realización en ensamblador |
|--|---|
| <pre> graph TD Entry(()) --> Cond{Si Cond} Cond -- Verdadero --> AccV[Acciones Verdaderas] Cond -- Falso --> AccF[Acciones Falsas] AccV --> Join(()) AccF --> Join Join --> Exit(()) </pre> | <pre> ; Con el uso de la instrucción CP, ; instrucciones de salto condicional BRNE, ; e instrucciones de salto relativo RJMP ; Por ejemplo, si la condición es R2 == R3 CP R2,R3 ;comparación de R2 y R3 BRNE ETI_ELSE ;Si R2 != R3 salta ETI_ELSE ADD R0, R3 ;Acciones verdaderas SUB R0, R5 INC R10 RJMP FIN_SI ;Salta al final de la estructura ETI_ELSE: SUB R0, R3 ;Acciones para el caso falso ADD R0, R5 DEC R10 FIN_SI: ;Fin de la estructura if </pre> |

3.2. Realización en ensamblador de la estructura de repetición.

La estructura de repetición permite ejecutar un bloque de instrucciones un número determinada de veces hasta que una condición ya no se cumpla. Existen tres tipos de estructuras de decisión: *WHILE*, *FOR* y *DO-WHILE*. En las siguientes tablas se muestra la forma de codificar en lenguaje ensamblador estas estructuras de repetición.

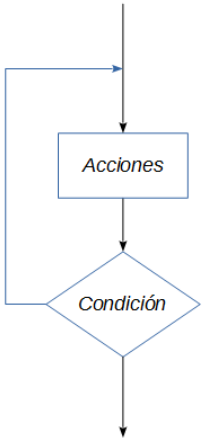
Tabla 3.3. Codificación de ensamblador de la estructura de control WHILE

| Estructura WHILE | Realización en ensamblador |
|---|--|
| <pre> graph TD Entry(()) --> Cond{Si Cond} Cond -- Verdadero --> Acciones[Acciones Verdaderas] Acciones --> Entry Cond -- Falso --> Exit(()) </pre> | <p>; Se ejecuta un grupo de acciones mientras ; se cumple una condición ; Por ejemplo, mientras se cumpla $R2 == R3$, ; se ejecuta el bloque de instrucciones</p> <p>INICIO_WHILE: CP R2,R3 ;Compara R2 y R3 BRNE FIN_WHILE ;Salta a FIN_WHILE ; si $R2 \neq R3$</p> <p>ADD R0, R3 ;instrucciones a repetir SUB R0, R5 INC R10</p> <p>RJMP INICIO_WHILE ; Salta al inicio</p> <p>FIN_WHILE:</p> |

Tabla 3.3. Codificación de ensamblador de la estructura de control FOR

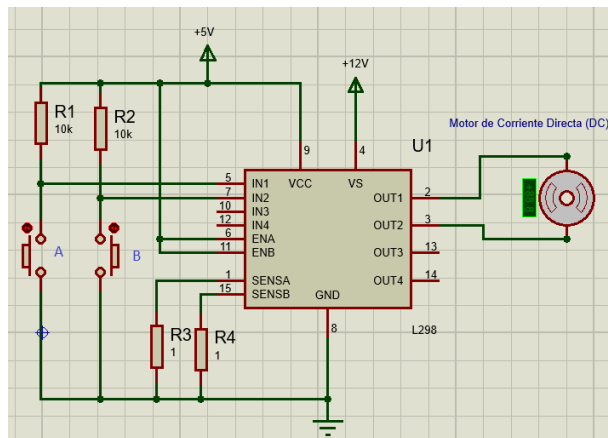
| Estructura FOR | Realización en ensamblador |
|---|---|
| <pre> graph TD Entry(()) --> Init[Con = inicio] Init --> Cond{Si Cond} Cond -- Verdadero --> Acciones[Acciones Verdaderas] Acciones --> Inc[Cont = Cont + 1] Inc --> Cond Cond -- Falso --> Exit(()) </pre> | <p>; Se establece el valor final al registro R16 LDI R16, valorFinal ; $R16 \leftarrow \text{valorFinal}$</p> <p>CLR R2 ; $R2 \leftarrow 0$ (contador)</p> <p>INICIO_FOR: CP R2,R16 ; compara el contador con el valor final BREQ FIN_FOR ; Salta a FIN_FOR cuando ; el contador sea igual al valor final</p> <p>ADD R0, R3 ;instrucciones a repetir SUB R0, R5 INC R10</p> <p>; Se incrementa el contador INC R2 ; contador \leftarrow contador +1</p> <p>RJMP INICIO_FOR ; Salta al inicio</p> <p>FIN_FOR:</p> |

Tabla 3.4. Codificación de ensamblador de la estructura de control DO-WHILE

| Estructura DO-WHILE | Realización en ensamblador |
|---|---|
|  | <p>INICIO_DO: ; Etiqueta que marca el inicio DO</p> <p>; Se escriben las instrucciones contenidas en ; la estructura DO-WHILE</p> <p>ADD R0, R3 SUB R0, R5 INC R10</p> <p>; Se realiza la condición DO-WHILE</p> <p>CP R2,R3 ;compara el registro R2 con R3 BREQ INICIO_DO ;mientras sean iguales R2 == R3 ;salta a inicio</p> |

Desarrollo

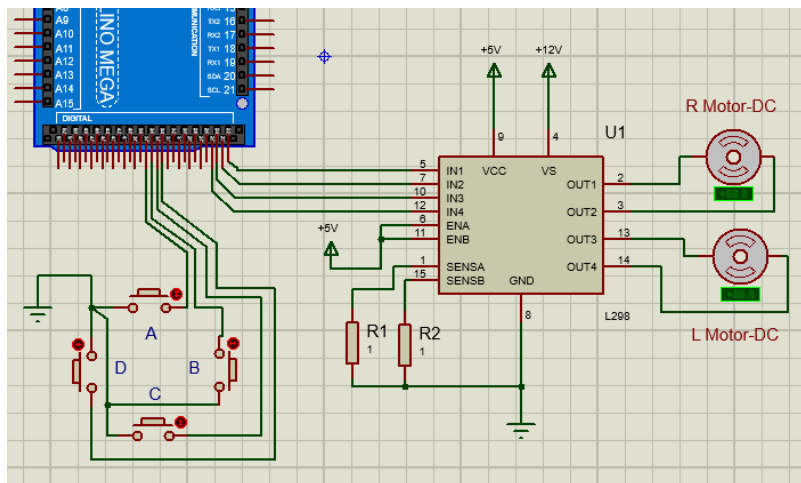
1. Cree un nuevo proyecto en Proteus y agregue la siguiente lista de componentes.
 - Motor de corriente continua (Motor-DC)
 - Motor de pasos (Motor-Bistepper)
 - Puente H Driver L298
 - Botón push-buton.
 - Resistencia
 - Simulino Mega
2. Elaborar en Proteus el siguiente circuito con el puente-H L298 para controlar un motor de corriente continua.



3. Realizar la simulación del circuito y hacer las pruebas que indican la tabla, en base al resultado, llenar los espacios que faltan en la tabla.

| Estado de los botones | | Entradas lógicas del L298 | | Descripción del estado del motor |
|-----------------------|--------------------|---------------------------|-----|----------------------------------|
| Botón A | Botón B | IN1 | IN2 | |
| <i>Sin apretar</i> | <i>Sin apretar</i> | | | |
| <i>Sin apretar</i> | <i>Apretado</i> | | | |
| <i>Apretado</i> | <i>Sin apretar</i> | | | |
| <i>Apretado</i> | <i>Apretado</i> | | | |

4. En Proteus crear el siguiente circuito con la tarjeta Arduino Atemega2560, push-button y dos motores de DC. Las siguientes tablas indican los puertos usados en la tarjeta Arduino.



| Puerto A (Salida) | Num. del Conector | Circuito L298 |
|-------------------|-------------------|---------------|
| Pin0 | 22 | 5 (IN1) |
| Pin1 | 23 | 7 (IN2) |
| Pin2 | 24 | 10 (IN3) |
| Pin3 | 25 | 12 (IN4) |

| Puerto C (Entrada) | Num. del Conector | Botón |
|-----------------------|----------------------|-------|
| Pin0 | 37 | A |
| Pin1 | 36 | B |
| Pin2 | 35 | C |
| Pin3 | 34 | D |

5. Con la información obtenida en el ejercicio 3, elaborar un programa en lenguaje ensamblador para el Atmega2560 que permita mover los motores de acuerdo a la siguiente tabla. NOTA. En el puerto C es necesario activar la resistencia *Pull-Up* interna.

| Estado de los botones | Estado del motor |
|--|--|
| A : Sin apretar B : Sin apretar C : Sin apretar D : Sin apretar | Motor R: Sin mover Motor L: Sin mover |
| A : Apretado B : Sin apretar C : Sin apretar D : Sin apretar | Motor R: Mover en sentido antihorario Motor L: Mover en sentido antihorario |
| A : Sin apretar B : Apretado C : Sin apretar D : Sin apretar | Motor R: Mover en sentido horario Motor L: Mover en sentido antihorario |
| A : Sin apretar B : Sin apretar C : Apretado D : Sin apretar | Motor R: Mover en sentido horario Motor L: Mover en sentido horario |
| A : Sin apretar B : Sin apretar C : Sin apretar D : Apretado | Motor R: Mover en sentido antihorario Motor L: Mover en sentido horario |

Bibliografía

- *Atmel 2549 8-bit AVR Microcontroller ATmega640/1280/1281/2560/2561 datasheet*
- *Atmel 0856 AVR Instruction Set Manual*
- *Atmel 42167 Atmel Studio User Guide*
- *Arduino mega2560 R3 diagrama esquemático.*
- *Avrdude 6.3 manual de usuario*
- *Hoja de especificaciones del circuito Dual Full Bridge Driver L298*