Principios de Mecatrónica – SDI-11561
Ingeniería en Mecatrónica

Hugo Rodríguez Cortés

Departamento de Ingeniería Eléctrica y Electrónica

Instituto Tecnológico Autónomo de México

Agosto 2023

Registro de estatus

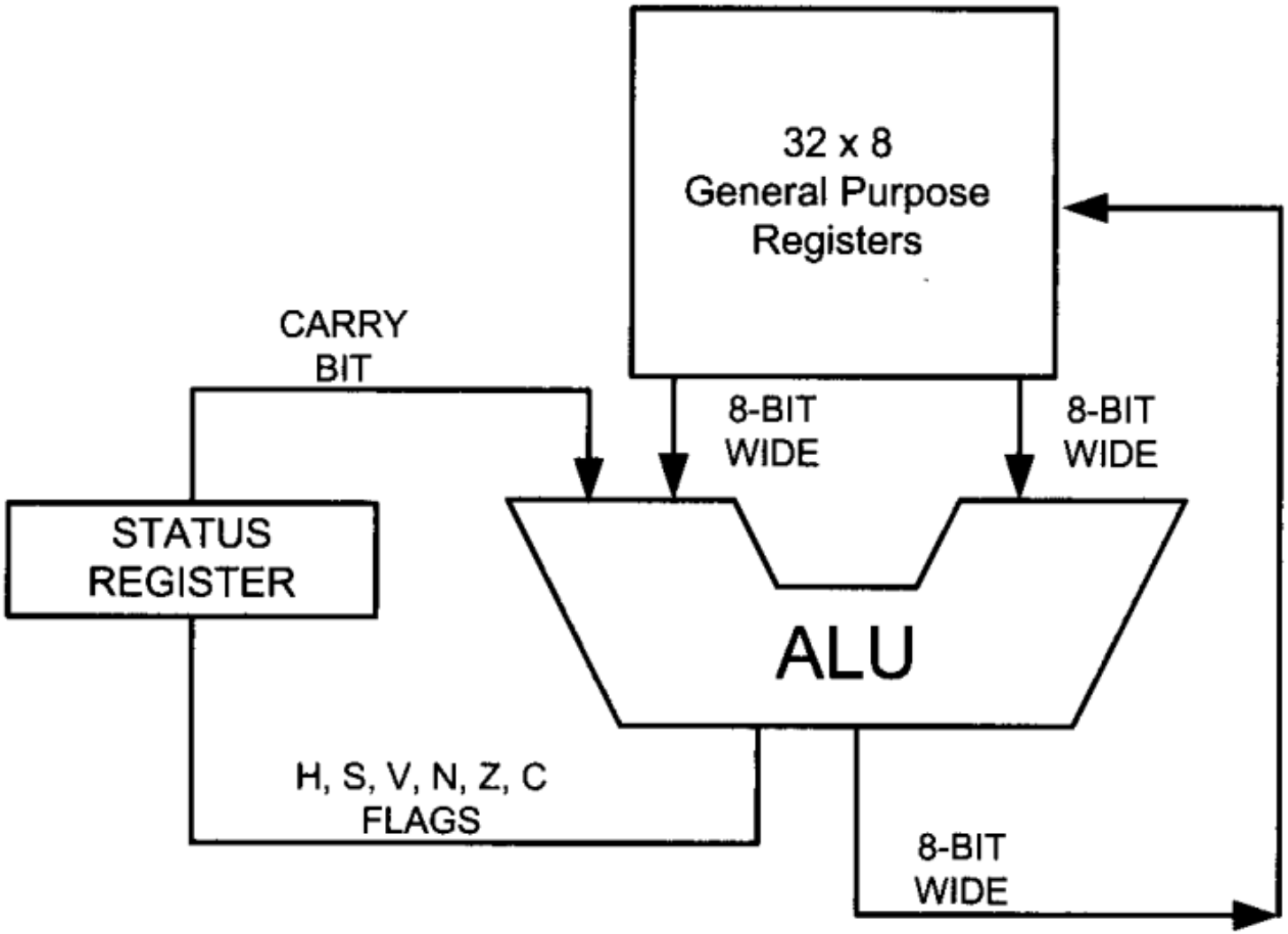
- Determine el valor de R21 después de las siguientes instrucciones

```
LDI   R21, 0×F5   ;   –
LDI   R22, 0×0B   ;   –
ADD   R21, R22    ;   –
```

- La suma en binario es

$$\begin{array}{r} 11110101 \\ +\ 00001011 \\ \hline \end{array}$$

ALU


32 x 8
General Purpose
Registers
CARRY
BIT
8-BIT
WIDE
8-BIT
WIDE
STATUS
REGISTER
ALU
H, S, V, N, Z, C
FLAGS
8-BIT
WIDE

Registro de estatus

- El microcontrolador AVR tiene un registro para indicar la condición aritmética de una operación. Este registro se conoce como registro de estatus (SReg).

- El registro de estatus tiene la siguiente estructura

Bit	D7	D6	D5	D4	D3	D2	D1	D0
SREG	I	T	H	S	V	N	Z	C

  - C (carry flag). C=1, si lleva 1 en el bit D7.
  - Z (zero flag). Z=1 si el resultado de la operación es igual a cero.
  - N (negative flag). Se utiliza para hacer operaciones con signo, si D7 = 0, número positivo, N = 0, si D7 =1, N=1, número negativo.

Registro de estatus

Bit	D7	D6	D5	D4	D3	D2	D1	D0
SREG	I	T	H	S	V	N	Z	C

- V (overflow flag). V $= 1$ si el resultado de una operación de números con signo causa un desbordamiento hacia el bit que indica el signo.
- S (sign bit). Es el resultado de una OR-exclusiva (XOR) entre las banderas N y V.
- H (half carry flag) H=1 si durante ADD o SUB se lleva 1 de D3 a D4.
- Las banderas I y T se analizarán después.

Registro de estatus (ADD)

- Determine el estatus de la bandera Z en cada línea de la siguiente secuencia

```
LDI   R20, 4   ;   –
DEC    R20     ;   –
DEC    R20     ;   –
DEC    R20     ;   –
DEC    R20     ;   –
```

En operaciones sin signo, la operación ADD puede modificar las banderas C, H y Z.

- Determine el valor de las banderas C, H y Z en las siguientes operaciones

```
LDI   R16, 0 × 38   ;   –
LDI   R17, 0 × 2F   ;   –
ADD    R16, R17     ;   –
```

Registro de estatus(ADD)

- Determine el valor de las banderas C, H y Z en las siguientes operaciones

```
LDI   R20, 0x9C   ;   –
LDI   R21, 0x64   ;   –
ADD   R20, R21    ;   –
```

```
LDI   R20, 0x88   ;   –
LDI   R21, 0x93   ;   –
ADD   R20, R21    ;   –
```

Registro de Estatus y Condicionales

Se cuenta con una serie de instrucciones de que realizan un salto condicional (branch) en función del valor de algunos bits del registro de estatus.

Instrucción	Resultado
BRLO	Branch si C=1
BRSH	Branch si C=0
BREQ	Branch si Z=1
BRNE	Branch si Z=0
BRMI	Branch si N=1
BRPL	Branch si N=0
BRVS	Branch si V=1
BRVC	Branch si V=0

Formato de datos

Se tienen cuatro formas para representar los bytes de datos en el lenguaje ensamblador para AVR.

- Números hexadecimales. Se utiliza $0\times$ ó $\$$ frente al número. $0 \times 99 = \$99$.

- Números binarios. Se utiliza 0b ó 0B frente al número. $0\text{b}00100101 = 0\text{B}00100101$.

- Números decimales. Se utiliza el número decimal sin índicadores. LDI $R17, 12$.

- Caracteres ASCII. Se utiliza un apóstrofe simple. $'2' = 0\text{b}00110010 = 0 \times 32 = \$32$.

Directivas en ensamblador

Las directivas dan instrucciones al ensamblador, no al CPU. Se conocen como pseudo-instrucciones. Las directivas ayudan a desarrollar el programa y hacerlo legible. No generan código de máquina.

- .EQU Esta directiva permite fijar valores o direcciones constantes

```
.EQU COUNT = 0 × 25      ; COUNT toma el valor de 0 × 25.
  ⋮                      ⋮
LDI   R21, COUNT         ; R21 = COUNT = 0 × 25
```

- .SET Se utiliza para definir un valor constante o una dirección fija. El valor asignado por .SET es modificable.
- .ORG Se utiliza para indicar el inicio de la dirección, para código y datos.
- .INCLUDE Agrega el contenido de un archivo al programa.

Directivas en ensamblador

- Ejemplo

```
.INCLUDE "m256def.inc"      ; Incluye el archivo m256def.inc
.ORG   0 × 00               ; Almacenar código de máquina
                            a partir de la dirección 0 × 00
```

- .DEVICE Define el microcontrolador que va a utilizarse.

```
.DEVICE ATMega2560      ; ATMega2560 designación del chip.
```

- Ejemplo

```
.EQU SUM = 0x120      ; Define SUM = 0x120.
LDI   R20, 5          ; cargar el 5 a R20
LDI   R21, 2          ; cargar el 2 a R21
ADD    R20, R21       ; R20 = R20 + R21
STS   SUM, R20        ; almacenar R20 en la dirección 0x120
```

Lazos en el AVR

El AVR tiene instrucciones para realizar saltos condicionales y no condicionales.

- Se llama lazo a la repetición de una secuencia de instrucciones ó de una instrucción un cierto número de veces, por ejemplo

```
LDI   R16, 0      ;  –
LDI   R17, 3      ;  –
ADD   R16, R17    ;  –
ADD   R16, R17    ;  –
ADD   R16, R17    ;  –
ADD   R16, R17    ;  –
ADD   R16, R17    ;  –
ADD   R16, R17    ;  –
```

Lazos en el AVR

- BRNE (branch if not equal) utiliza la bandera Z del registro de estatus.

```
BACK:   ...          ; inicio del lazo.
        ...          ; instrucción del lazo.
        ...          ; instrucción del lazo.
        DEC   Rn     ; decrementar Rn, Z=1 si Rn = 0.
        BRNE  BACK   ; regresar a BACK si Z=0.
```

- Ejemplo. Realizar un programa para multiplicar 3 por 10 y escribir el resultado en PORT B.

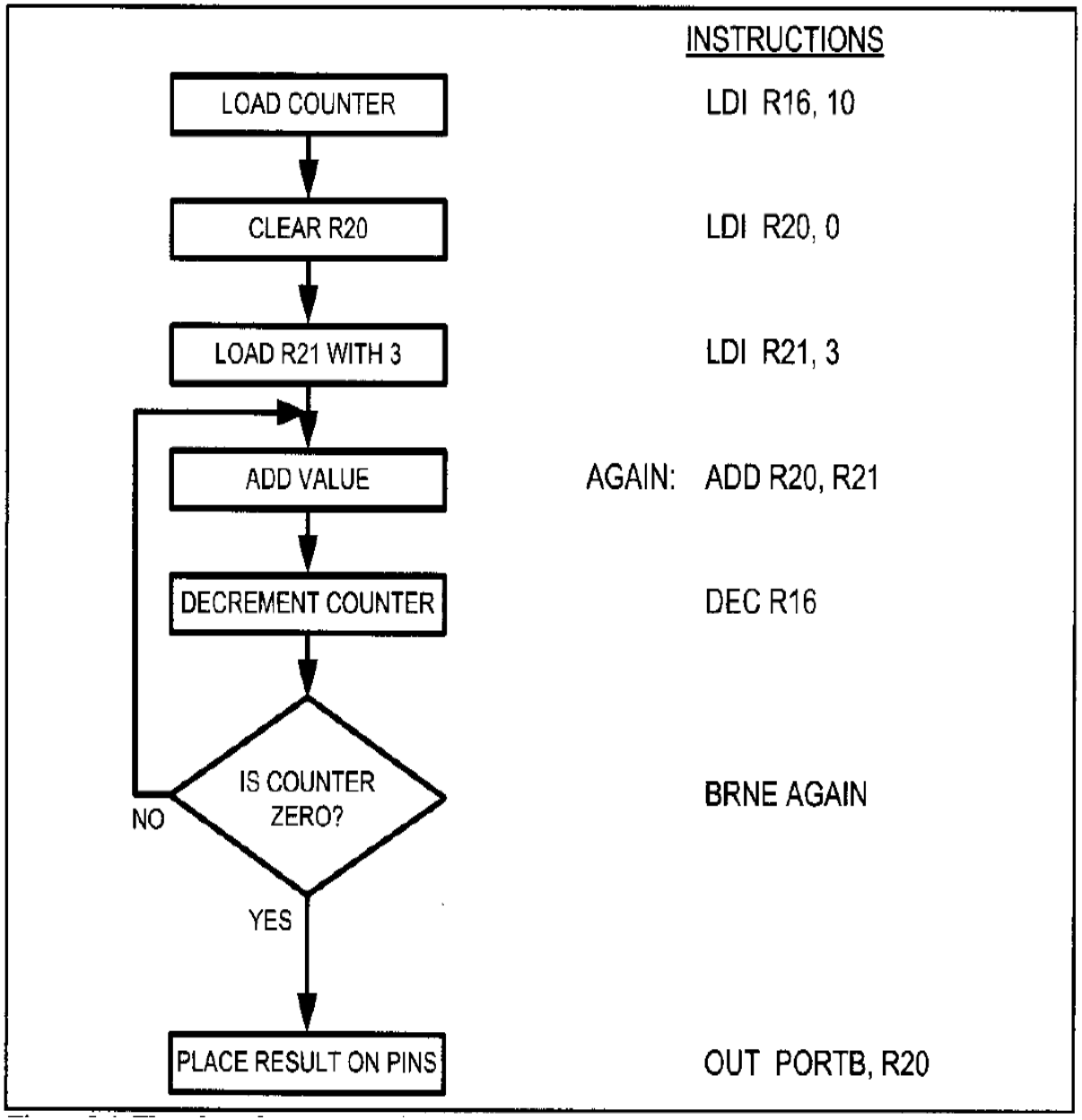
Lazos en el AVR

- Solución

```
.INCLUDE  "M32DEF.INC"        ;  –
          LDI   R16, 10       ;  –
          LDI   R20, 0        ;  –
          LDI   R21, 3        ;  –
AGAIN:    ADD   R20, R21      ;  –
          DEC   R16           ;  –
          BRNE  AGAIN         ;  –
          OUT   PORTB, R20    ;  –
```

- ¿Cuál es el máximo número de veces que puede repetirse un conjunto de instrucciones?

Lazos en el AVR

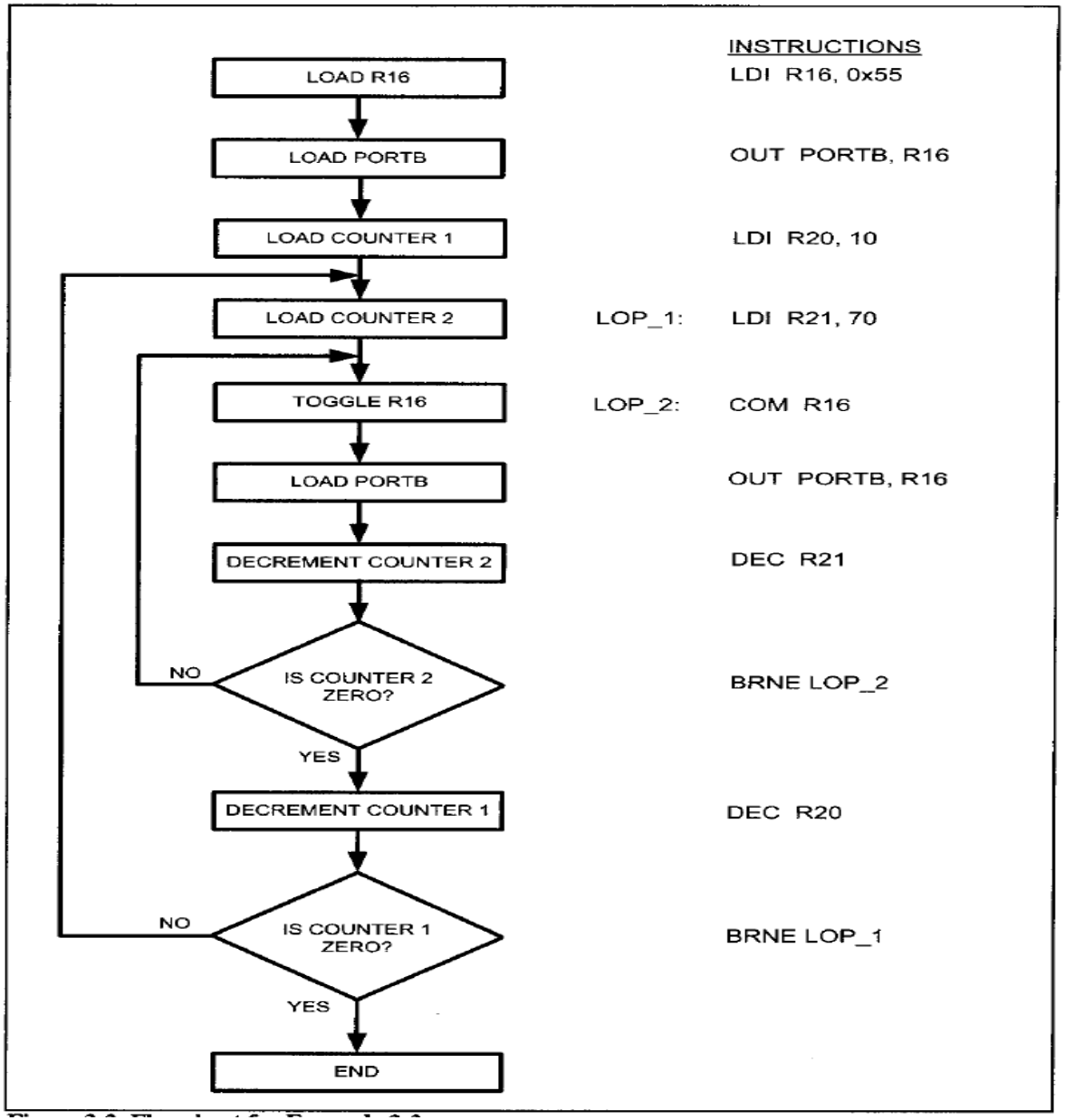


Lazo dentro de otro lazo

- Ejemplo. El siguiente programa carga al registro PORTB el valor de 0x55 y su complemento 700 veces.

```
.INCLUDE  "M32DEF.INC"   ;                              –
.ORG      0x00           ;                              –
          LDI    R16, 0 × 55   ;                               –
          OUT    PORTB, R16    ;                               –
          LDI    R20, 10       ;                               –
LOP_1:    LDI    R21, 70       ;                               –
LOP_2:    COM    R16           ;                               –
          OUT    PORTB, R16    ;                               –
          DEC    R21           ;                               –
          BRNE   LOP_2         ;                               –
          DEC    R20           ;                               –
          BRNE   LOP_1         ;                               –
```

Lazo dentro de un lazo



Otros saltos condicionales

- BREQ (brach if equal, branch if Z = 1) Ejemplo

```
OVER:  IN     R20, PINB  ;                              –
       TST    R20        ;  TST verifica si R20 = 0 ⇒ Z=1
       BREQ   OVER       ;                              –
```

El programa sale del lazo cuando PINB toma un valor diferente de cero. TST también asigna N = 1 si D7 = 1.

- BRSH (branch if same or higher, branch if C = 0) Al ejecutar "BRSH label" el CPU verifica el registro de estatus. Si C = 0 el CPU comienza a recolectar y ejecutar las instrucciones del label. Si C = 1 el programa no salta continuando con el programa.

Otros saltos condicionales

- Ejemplo. Sumar 0x79 + 0xF5 + 0xE2 colocando el resultado en dos bytes.

```
.INCLUDE   "M32DEF.INC"   ;                              –
.ORG       0x00           ;                              –
           LDI    R21, 0          ;                        –
           LDI    R20, 0          ;                        –
           LDI    R16, 0x79       ;                        –
           ADD    R20, R16        ;                        –
           BRSH   N_1             ;                        –
           INC    R21             ;                        –
N_1:       LDI    R16, 0xF5       ;                        –
           ADD    R20, R16        ;                        –
           BRSH   N_2             ;                        –
           INC    R21             ;                        –
N_2:       LDI    R16, 0xE2       ;                        –
           ADD    R20, R16        ;                        –
           BRSH   OVER            ;                        –
           INC    R21             ;                        –
OVER:
```

Saltos no condicionales

- JMP "label" Salta a cualquier dirección dentro de la memoria del programa. No está disponible en todos los dispositivos debido a que algunos tienen una capacidad de memoria limitada. Ejemplo

```
        MOV   R21, R20   ;                    –
        JMP   OVER       ;                    –
        DEC   R21        ;                    –
OVER:
        RJMP  OVER       ;                    –
```

- RJMP "label" Salto no condicional disponible en dispositivos con baja capacidad de memoria. Las direcciones relativas tienen un rango entre 0×0 y $0\times$FFF. Se prefiere sobre JMP ya que ocupa menos espacio de memoria.

Ubicación del código en ROM

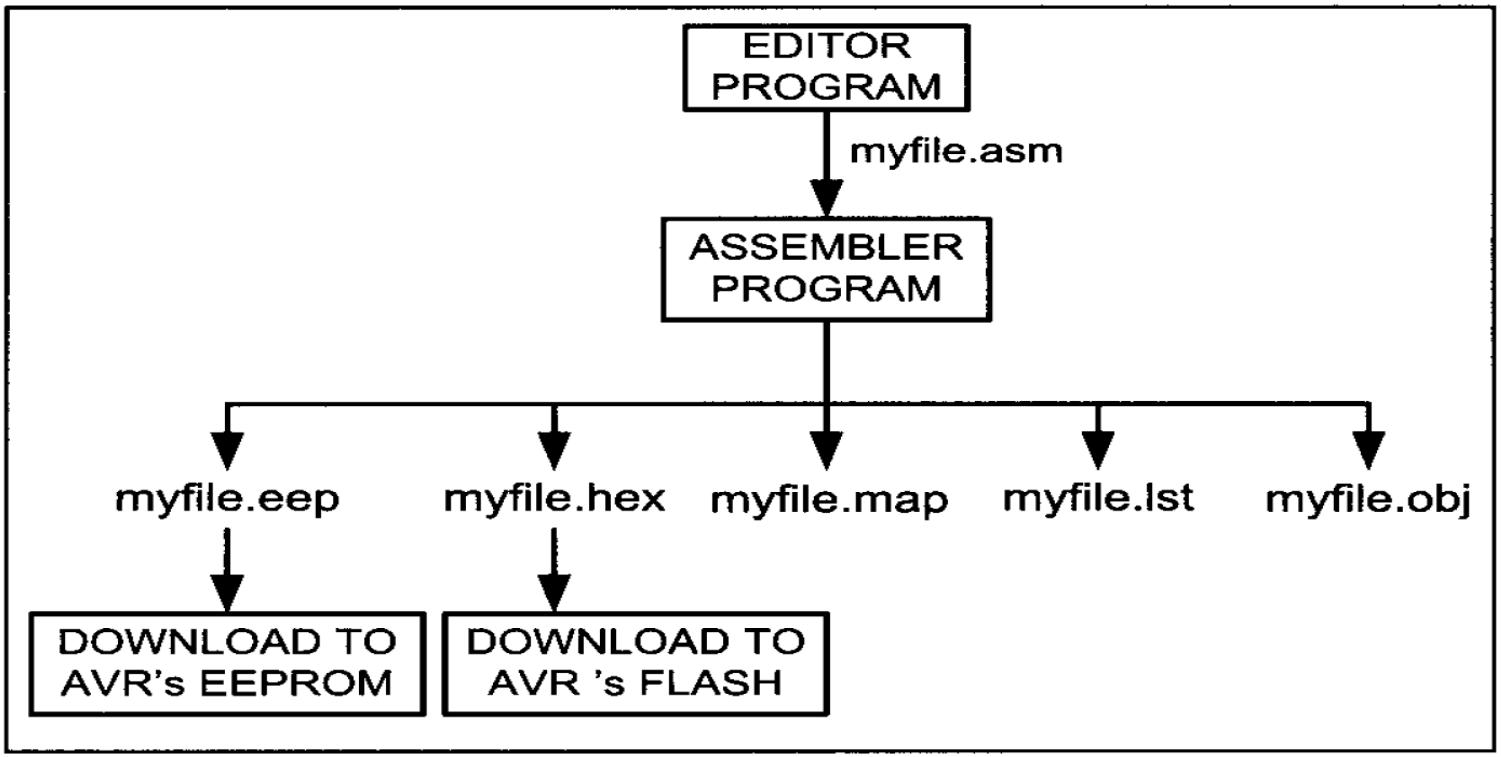
- Cuando el AVR se prende o se resetea, el CPU inicia en la localidad de memoria 0000. Por lo que el contador de programa[a] tiene el valor 0000. Esto indica que el primer código de operación esta almacenado en la dirección de memoria 0000.
- Considere el siguiente programa

```
      ;AVR Assembly Language Program To Add Some Data.
      ;store SUM in SRAM location 0x300.

      .EQU  SUM   = 0x300      ;SRAM loc $300 for SUM

      .ORG 00                  ;start at address 0
      LDI R16, 0x25            ;R16 = 0x25
      LDI R17, $34             ;R17 = 0x34
      LDI R18, 0b00110001      ;R18 = 0x31
      ADD R16, R17             ;add R17 to R16
      ADD R16, R18             ;add R18 to R16
      LDI R17, 11              ;R17 = 0x0B
      ADD R16, R17             ;add R17 to R16
      STS SUM, R16             ;save the SUM in loc $300
HERE: JMP HERE                 ;stay here forever
```

[a] Registro que indica al CPU la dirección de la instrucción que debe ejecutarse.

Ubicación del código en ROM

El editor del programa realiza la compilación y genera los archivos que deben cargarse a la memoria ROM del microcontrolador.



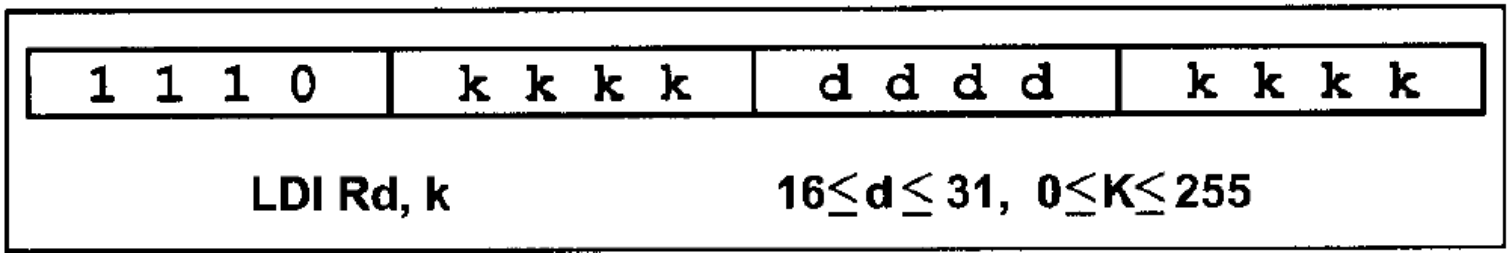
Ubicación del código en ROM

Una parte del código fuente tiene la siguiente estructura

```
AVRASM ver. 2.1.2  F:\AVR\Sample\Sample.asm Tue Mar 11 11:28:34 2008

                ;store SUM in SRAM location 0x300.
                   .DEVICE ATMega32
                   .EQU  SUM    = 0x300      ;SRAM loc $300 for SUM

                   .ORG 00                   ;start at address 0
000000 e205        LDI R16, 0x25             ;R16 = 0x25
000001 e314        LDI R17, $34              ;R17 = 0x34
000002 e321        LDI R18, 0b00110001       ;R18 = 0x31
000003 0f01        ADD R16, R17              ;add R17 to R16
000004 0f02        ADD R16, R18              ;add R18 to R16
000005 e01b        LDI R17, 11               ;R17 = 0x0B
000006 0f01        ADD R16, R17              ;add R17 to R16
000007 9300 0300   STS SUM, R16              ;save the SUM in loc $300
000009 940c 0009 HERE: JMP HERE              ;stay here forever

RESOURCE USE INFORMATION
------------------------
...
Memory use summary [bytes]:
Segment    Begin     End       Code    Data    Used     Size    Use%
-------------------------------------------------------------------
[.cseg] 0x000000 0x000016      22       0      22 unknown       -
[.dseg] 0x000060 0x000060       0       0       0 unknown       -
[.eseg] 0x000000 0x000000       0       0       0 unknown       -

Assembly complete, 0 errors, 0 warnings
```

Ubicación del código en ROM

- La lista muestra que en la dirección 0000 se almacena el código operacional E205. Esto es, la instrucción LDI R16, 0x25 tiene el código operacional E205. El código operacional para LDI Rd, k tiene tiene la estructura siguiente



- En la lista se tienen los siguientes códigos operacionales