

Una introducción asm

Empezando

Ensamblador es un lenguaje de bajo nivel. Consiste en una lista de instrucciones que son de ninguna manera comparable a cualquier cosa que pueda saber de C, Basic o Pascal. El AVR tiene alrededor de 120 de ellos, dependiendo del tipo y de sus periféricos y tamaño. Usted puede descargar el juego de instrucciones del sitio web de Atmel e imprimir el resumen (una lista de todas las instrucciones y qué tipo de operandos que tienen). Si lo descarga, sugiero imprimir las páginas 1 y 10 a 15 Eso no es mucho y sin embargo es todo lo que necesita para empezar. He aquí un enlace a la misma. El documento es de aproximadamente 150 páginas en total.

Vamos a echar vistazo a probablemente el programa más fácil posible:

```
main:                                ; esta es una de la etiqueta "principal"
```

```
    rjmp main                        ; Salto relativo a la principal
```

"Principal" (la primera línea) no se traduce por el ensamblador, sino que se utiliza como una etiqueta. Esta etiqueta sustituye a una dirección en el espacio de código (memoria FLASH) del AVR. Exactamente a esta dirección se coloca la siguiente instrucción (rjmp principal). Cuando se ejecutan a lo largo de la instrucción, la CPU salta al "principal" de nuevo. El salto se repetirá una y otra vez, lo que resulta en un bucle infinito.

La tabla de vectores de interrupción puede ser utilizado por usted para decirle al micro lo que tiene que hacer cuando se produce una interrupción específica. Los AVR's normales tienen espacio para una instrucción por vector de interrupción (un rjmp por ejemplo). Esta instrucción se ejecutará cuando se produzca la interrupción (No hay más que decir sobre esto, pero no ahora ...)

El primer vector de interrupción es el "vector de reset". Contiene la instrucción que el CPU debe ejecutar cuando se produce un reset. Lo utilizaremos para ir a nuestro programa ya teníamos anteriormente:

```
.org 0x0000          ; la siguiente instrucción se debe escribir
                      para abordar 0x0000

rjmp main            ; el vector de reset: saltar al "principal"

                      ;

main:                ; esta es la principal etiqueta ""

rjmp main            ; Salto relativo a la principal
```

Suponiendo que nuestro AVR está funcionando a 4 MHz (4 millones de ciclos de reloj por segundo), ¿cuánto tiempo lleva todo esto? RAV son bastante rápido - la mayoría de las instrucciones se pueden ejecutar en uno o dos ciclos de reloj. Algunas instrucciones necesitan un tiempo de poco más, pero estos no son importantes ahora. A medida que el reloj externo no está dividido internamente (algunos otros microcontroladores hacen que, al igual que el HC11 de Motorola, pero eso es una vieja), dos ciclos de reloj de 4 MHz significa que la instrucción se lleva a 0,0000005 segundos. Bastante rápido!

Suponiendo que nuestro AVR está funcionando a 4 MHz (4 millones de ciclos de reloj por segundo), ¿cuánto tiempo lleva todo esto? RAV son bastante rápido - la mayoría de las instrucciones se pueden ejecutar en uno o dos ciclos de reloj. Algunas instrucciones necesitan un tiempo de poco más, pero estos no son importantes ahora. A medida que el reloj externo no está dividido internamente (algunos otros microcontroladores hacen que, al igual que el HC11 de Motorola, pero eso es una vieja), dos ciclos de reloj de 4 MHz significa que la instrucción se lleva a 0,0000005 segundos. Bastante rápido!

La primera cosa que hice cuando empecé en AVR's estaba haciendo un flash LED. I/O Ports).

onmouseover="this.style.backgroundColor='#e6ff99'"

onmouseout="this.style.backgroundColor='#fff'">Los LEDs pueden ser conectados a puertos I / O del AVR

(Arquitectura -> Puertos E / S). Estos se pueden establecer como entrada o salida de forma individual para cada pin y también se puede habilitar una resistencia pull-up interna si el pin del puerto se fija a la entrada. Cada puerto I / O tiene tres registros que puede trabajar: registro de datos del puerto (por ejemplo PortB), la Dirección de Registro de datos (por ejemplo DDRB) y el registro de Pin (por ejemplo PinB). Para configurar un pin como un pin de salida, establezca su correspondiente bit en el registro de dirección de datos. El valor de salida (0 ó 1), entonces se puede establecer en el registro Puerto de datos.

Si usted tiene una STK500, consígase un ATmega8 en un paquete DIP y un cristal de 4 MHz. La lata micro conectado a la toma de destino verde llamado "SCKT3200A2".

Conecte el cable de 6 pines de "ISP6PIN" al jefe de la ISP para el socket verde. Ese es el llamado "SPROG2". El valor inicial del sistema oscilador puente predeterminado es el oscilador de software. Queremos utilizar el oscilador de cristal en su lugar. Ajuste el "OSCSEL" puente para cerrar los pines 2 y 3 (el pin 1 está marcado con un "1").

"Vobjetivo", "AREF", "Reset" y "XTAL1" debe ser cerrado.

Por supuesto, el mega8 debe ser el único micro conectados a la STK. No inserte más de un AVR a la vez! Desde aquí, tomar uno de los dos cables-cables y utilizarlo para conectarse "PB3" (que es una de las clavijas "PORTB") a "LED0" (que es uno de los "LEDS" alfileres). El cristal pertenece a la toma de cristal en el STK500.

Los LEDs STK500 están conectados al AVR a través de algunos componentes adicionales. Si lo desea, puede echar un vistazo en el circuito de LED en la documentación STK500 (se incluye en el archivo de ayuda AVR Studio y también vino con el STK). El hecho más importante es que el LED está conectado a ser "activa baja", lo que significa que si PortB.3 es baja ahora (nos conectamos a uno de los LED), el LED se encenderá.

Si usted no tiene un STK, conectar el LED a PortB.3 través de una resistencia de limitación de corriente (alrededor de 470 Ohms está bien, el valor no es crítico). Conecte la resistencia al pasador del puerto y el cátodo del LED, el ánodo va a Vcc. Esto dará lugar a la misma conducta "activa baja".

Ahora vamos a cambiar nuestro programa un poco, así que configura todos los pines PORTB como pines de salida. Después de un reset, todos los bits de datos se ponen a cero, por lo que el LED debe estar encendido cuando se ejecuta el programa:

```
.org 0x0000          ; la siguiente instrucción se debe escribir
                      ; para abordar 0x0000

rjmp main            ; el vector de reset: saltar al "principal"

main:                ; esta es la etiqueta "main"
ldi r16, 0xFF         ; registro de carga 16 con 0xFF
                      ; (todos los bits son 1)
out DDRB, r16         ; escribir el valor en r16 (0xFF) a la
                      ; Dirección de registros de datos B

loop:                ; esta es una nueva etiqueta que
                      ; utilizamos para un "bucle de no hacer
                      ; nada"

rjmp loop             ; saltar a bucle
```

Se inserta el nuevo bucle de manera que podamos establecer los bits de dirección de datos a nuestras necesidades y luego bucle sin hacer eso de nuevo. Podríamos, sin embargo, incluir las instrucciones de carga y almacenamiento (LDI y fuera) en el bucle. No estaría de más, pero los micro sería configurar los puertos en el mismo valor una y otra vez.

Si no lo ha hecho, descargue AVR Studio (yo prefiero la versión 3.5, no 4!) Del sitio web de Atmel e instalarlo. Ahora lee "Crear un proyecto" en la Sección de AVR Studio y cree un nuevo proyecto, elija su nombre favorito y tener en cuenta que el LED parpadeará cuando esté terminado:-)

Ahora es el momento para iniciar una nueva página (ya es suficiente). Nuestro código crecerá un poco ... que podría ser útil tener una calculadora para todas las cosas de tiempo ...

En la primera página que te dije cómo funciona un programa simple, cómo crear un proyecto y algunos detalles sobre el AVR. Ahora es el momento de echar un vistazo a lo que queremos hacer y cómo se puede hacer.

Queremos hacer un flash LED. Básicamente queremos cambiar por intervalos en un bucle. El LED está conectado a PortB.3, uno de los RAV pins E / S que se configura como una salida

Los bits individuales en el espacio de E / S se pueden borrar y establecer con el CBI (poco claro en I / O) y OSE (bit establecido en E / S) instrucciones. Por desgracia, esto no funciona para todos los registros de E / S. Abra el conjunto de instrucciones AVR (al completo, no sólo las 7 páginas que ha imprimido a cabo) y busque CBI: Puede sólo trozos claros en los registros 0-31 (Hex: 0x00 ... 0x1F). PortB está en este rango (0x18), por lo que podemos utilizar CBI (lo mismo para OSE).

A medida que el LED está en ON cuando PortB.3 es baja, el LED se puede encender con ICC y con OSE. Vamos a añadir que al bucle:

```

.org 0x0000      ; la siguiente instrucción se debe escribir
                  ; para abordar 0x0000
rjmp main        ; the reset vector: jump to "main"

main:            ; esta es la etiqueta "main"
di r16, 0xFF      ; registro de carga 16 con 0xFF (todos los
                  ; bits son 1)
out DDRB, r16     ; escribir el valor en r16 (0xFF) a la
                  ; Dirección de registros de datos B

loop:            ;
sbi PortB, 3      ; apagar el LED
cbi PortB, 3      ; encenderlo
rjmp loop         ; saltar a bucle

```

La razón por la que primero apagarlo es que ya estaba al entrar en el bucle por primera vez: Después de reset, todos los bits de datos del puerto son cero. Al establecer el bit 3 en DDRB configuramos portB.3 como salida y cambiamos el LED de estado

Ahora puede programar su mega8 con este código, pero sólo se vería el LED es encendido todo el tiempo. El bucle cambia el LED apagado (OSE PortB, 3), que tiene 1 ciclo de reloj. Entonces, después de 0.00000025 (nuevo 1 ciclo de reloj) segundos, el LED se ilumina de nuevo (CBI PortB, 3). El rjmp tarda 2 ciclos de reloj (0,0000005 segundos). Eso es un poco demasiado rápido para el ojo. Tenemos que perder algo de tiempo entre la conexión del LED encendido y apagado (digamos 0,5 segundos)

Este pequeño bucle lleva mucho tiempo: clr necesita 1 ciclo, adiw necesita dos ciclos y Brne necesita 2 ciclos si la sucursal se hace y 1 de otra forma. Cada vez que los registros no desborden el bucle de toma adiw (2) Brne (2) = 4 ciclos. Esto se hace 0xFFFF veces antes de que ocurra el desbordamiento. La próxima vez que el bucle sólo necesita 3 ciclos, porque ninguna rama se hace. Esto se suma a $4 * 0xFFFF$ (looping) 3 (desbordamiento) 2 (CLR) = 262.145 ciclos. Esto todavía no es suficiente: $2000000/262145 \sim 7.63$

Tenemos que ajustar el lazo un poco y también Creta un bucle "en torno a" lo que contendrá nuestro bucle 262,145 ciclo. Para afinar el bucle interior que tenemos que cambiar las instrucciones CLR para LDI para que podamos utilizar un valor inicial diferente de 0 El lazo "exterior" será de cuenta descendente de 8 a cero utilizando r16. Así es como el código de demora se ve ahora

ldi r16, 8	; r16 carga con 8
outer_loop:	; etiqueta bucle externo
	;
ldi r24, 0	; registro claro 24
ldi r25, 0	; registro claro 25
delay_loop:	; la etiqueta bucle
adiw r24, 1	; "Agregan inmediata a la palabra": r24: r25 se incrementan
brne delay_loop	; si no hay desbordamiento ("rama si no igual"), vuelve a "delay_loop"
	;
dec r16	; r16 decremento
brne outer_loop	; y bucle si bucle exterior no ha terminado

Una vez más, algunos cálculos: El bucle interior ahora se trata como una sola instrucción BIG necesitan 262.145 ciclos de reloj. LDI necesita 1 ciclo de reloj, diciembre también necesita 1 ciclo de reloj y Brne necesita 1 o 2 ciclos (véase más arriba). Las necesidades generales de bucle: $262145 \text{ (bucle interior)} + 1 \text{ (diciembre)} + 2 \text{ (Brne)} = 262148 * 8 = 2.097.184$ ciclos más la LDI = 2097185. Espere inicial. Resta un porque la última Brne no dio lugar a una rama, por lo que necesita 2097184 ciclos. Esto es más como lo que queremos, pero 97.184 ciclos demasiado largos. Aquí es donde la puesta a punto viene en - tenemos que cambiar el valor inicial de r24: r25

El bucle externo se ejecuta 8 veces e incluye el "circuito grande-interior-instrucción". Tenemos que restar algunos ciclos del bucle interno: $97184/8 = 12.148$ ciclos por bucle interior. Esto es lo que el bucle interior tiene que ser más corto. Cada iteración del bucle interno es de 4 ciclos (el último hace 3 pero eso no es tan importante), así que vamos a dividir los 12.148 por 4. Eso es 3036.5 o 3037 menos iteraciones. Este es nuestro nuevo valor de inicialización para r24: r25!

Ahora, si lo desea, hacer todos esos cálculos otra vez: El resultado es 2.000.000 ciclos de reloj! Ahora sólo hay que poner esto en una rutina separada y llamarlo desde el bucle principal LED parpadea:

.org 0x0000 ; la siguiente instrucción se debe escribir para
abordar 0x0000

```

rjmp main          ; el vector de reset: saltar al "principal"

```

```

main:
    ;

```

```

ldi r16, low(RAMEND)          ; configurar la pila
out SPL, r16                  ;
                               ;
ldi r16, high(RAMEND)
out SPH, r16                  ;
                               ;

ldi r16, 0xFF                 ; registro de carga 16 con 0xFF (todos los
                               bits son 1)
out DDRB, r16                 ; escribir el valor en r16 (0xFF) a la
                               Dirección de registros de datos B

loop:                          ;
sbi PortB, 3                  ; apagar el LED
rcall delay_05                ; esperar a que la mitad de un segundo
cbi PortB, 3                  ; encenderlo
rcall delay_05                ; esperar a que la mitad de un segundo
rjmp loop                     ; saltar a bucle
delay_05                      ; la subrutina:
                               ;
ldi r16, 8                    ; r16 carga con 8
outer_loop:                   ; etiqueta bucle externo
                               ;
ldi r24, low(3037)            ; registros de carga R24: r25 con 3037,
                               nuestro nuevo valor init
ldi r25, high(3037)           ;

delay_loop:                   ; la etiqueta bucle
adiw r24, 1                   ; "Agregan inmediata a la palabra": r24: r25
                               se incrementan
brne delay_loop               ; si no hay desbordamiento ("rama si no
                               igual"), vuelve a "delay_loop"
dec r16                       ; r16 decremento
                               ;
brne outer_loop               ; y bucle si bucle exterior no ha terminado
ret                           ; return from subroutine

```

Para tratar esto en el simulador o montarla, también agregar esta línea al principio del texto:

```
include "c:\avr_studio_working_directory\appnotes\m8def.inc"
```

Esto incluirá los ATmega8 archivo DEF de Atmel que incluye cosas como la definición de direcciones PortB. Sin este archivo el ensamblador escupa advertencias de error!