

Universidad Autónoma de Baja California



Microcontroladores y microprocesadores

Practica 6. Programación en Lenguaje Ensamblador del ATmega1280

Alumno: Caudillo Sanchez Diego

Matricula: 1249199

Docente: Jesus Adan Garcia Lopez

Fecha de entrega: 03 de abril de 2020

Teoría

Arquitectura interna y conjunto de instrucciones del ATmega1280

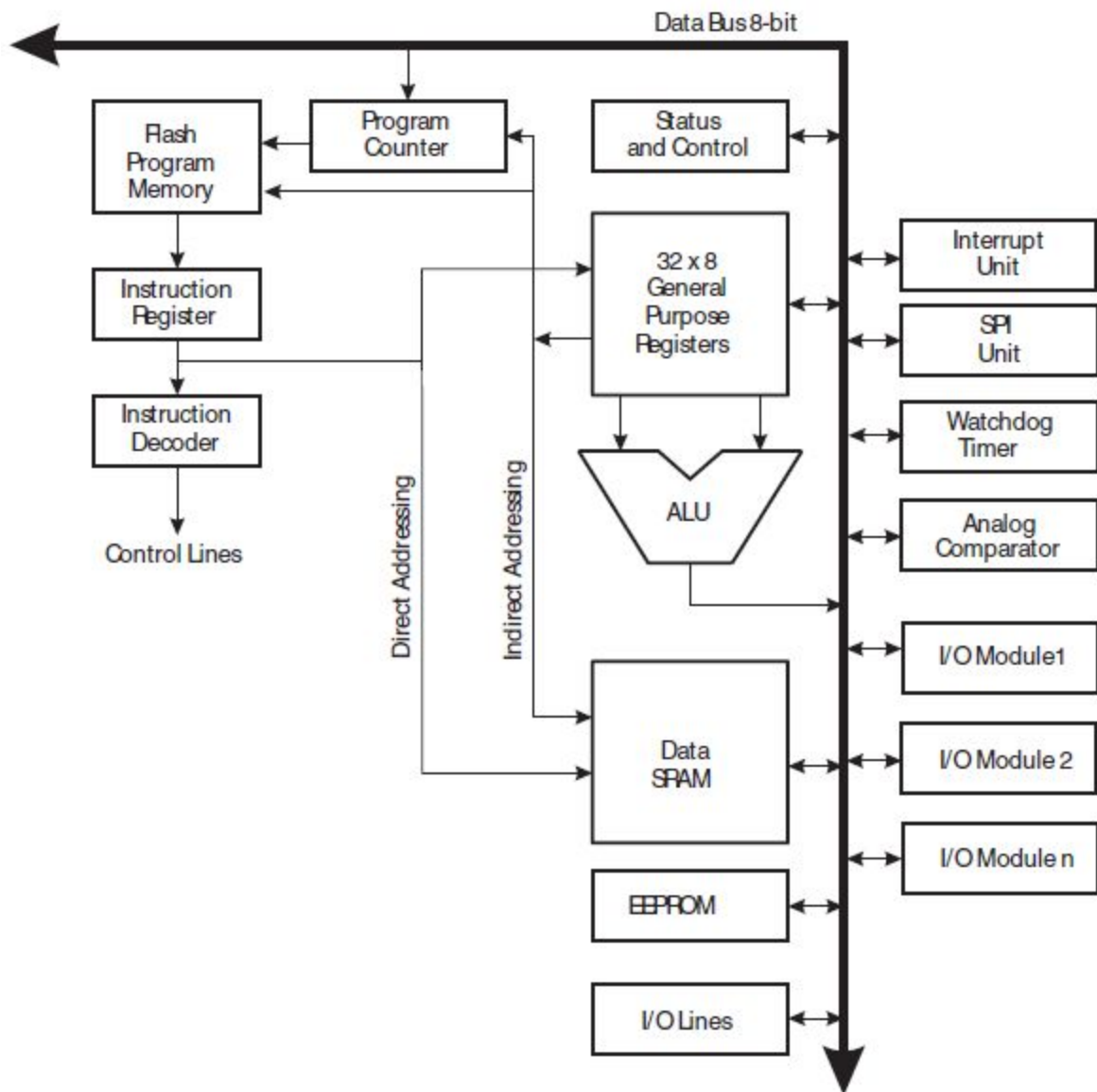


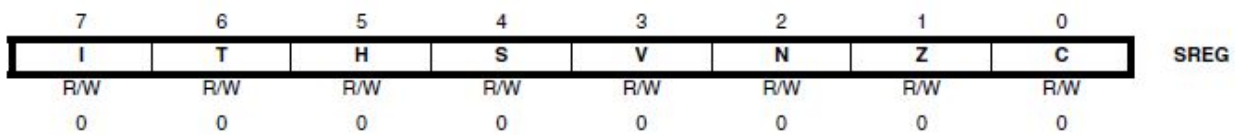
Figura 1. Arquitectura de 1280

ALU - Unidad Aritmetica y Logica

La ALU de alto rendimiento del AVR funciona en conexión directa con los 32 registros de trabajo de propósito general. Dentro de un solo ciclo de reloj, operaciones aritméticas entre registros de propósito general o entre un registro y un inmediatos son ejecutados. Las operaciones de ALU se dividen en tres categorías principales: funciones aritméticas, lógicas y de bits. Algunas implementaciones de la arquitectura también proporcionan un poderoso multiplicador que admite ambos multiplicación con signo / sin signo y formato fraccional.

Estados de registros

Contiene la información acerca de los resultados de la ejecución aritmética más reciente. Esta información se utiliza para alterar el curso del programa para ejecutar operaciones condicionales.



- **Bit 7 - I:** Habilitador de interrupciones globales
- **Bit 6 - T:** Almacenamiento de una copia de bit
- **Bit 5 - H:** Bandera de acarreo de nibble
- **Bit 4 - S:** Bit de signo
- **Bit 3 - V:** Bandera de desbordamiento, complemento 2
- **Bit 2 - N:** Bandera de signo negativo
- **Bit 1 - Z:** Bandera de zero
- **Bit 0 - C:** Bandera de acarreo

Registros de proposito general

Estos registros es una optimización de la arquitectura RISC mejorada del AVR, con el fin de alcanzar el rendimiento y flexibilidad requerida.

- Un operando con salida de 8-bits y el resultado con entrada de 8-bits.
- Dos operandos con salida de 8-bit y el resultado con entrada de 8-bits.
- Dos operandos con salida de 8-bits y un resultado con entrada de 8-bits.
- Un operando con salida de 16-bits y un resultado con entrada de 16-bits.

Stack Pointer

La pila o *stack* se usa principalmente para almacenar datos temporales, para almacenar variables locales y para almacenar direcciones de retorno después de interrupciones y llamadas de subrutina. El registro de puntero de pila siempre apunta a la parte superior de la pila. Tenga en cuenta que el Stack se implementa a medida que crece desde ubicaciones de memoria más altas a

ubicaciones de memoria más bajas. Esto implica que un el comando PUSH de pila disminuye el puntero de pila.

Generación de números pseudoaleatorios

Los números pseudo aleatorios son algoritmos que usan fórmulas matemáticas para producir secuencias de números aleatorios, de tal modo que los números se aproximan a las propiedades de los números aleatorios.

Los generadores de números pseudo aleatorios inician en un estado arbitrario usando una *estado de semilla*. Muchos números son generados en un corto tiempo y también pueden reproducirse después, su el punto inicial en la secuencia es conocida. Por lo tanto, los números son determinísticos y eficientes.

Uno de los algoritmos más comunes y más antiguos para generar números pseudo-aleatorios es el de *Generador Lineal Congruencial*, dada por la siguiente función.

$$X_{n+1} = (aX_n + c) \mod m$$

Donde X es la secuencia de valores pseudoaleatorios, y

$m, 0 < m$ Es el *modulo*.

$a, 0 < a < m$ Es el *multiplicador*.

$c, 0 \leq c < m$ Es el *incremento*.

$X_0, 0 \leq X_0 < m$ Es la *semilla* o *valor inicial*.

Se genera el siguiente entero aleatorio usando el número aleatorio anterior, el entero constante y el módulo. Para iniciar, el algoritmo necesita una semilla inicial, la cual puede ser obtenida por algunos medios. La apariencia de aleatoriedad se proporciona al realizar módulos aritméticos.

Lagged Fibonacci

Es otro ejemplo de generador de números pseudoaleatorios. El cual fue creado como una mejora del Generador Lineal Congruencial. El algoritmo se basa en la generalización de la secuencia Fibonacci.

La ecuación esta dada de la siguiente manera:

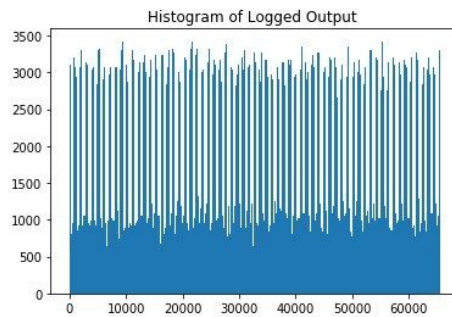
$$S_n \equiv S_{n-j} \star S_{n-k} \pmod{m}, 0 < j < k$$

En este caso, el nuevo término está dado por cualquiera de los dos términos anterior. M usualmente es una potencia de 2, normalmente 2^{32} o 2^{64} . El operador \star denota una operación binaria, como lo puede ser una suma, resta, multiplicación o XOR. La teoría de este generador es compleja, y puede no ser suficientemente simple para elegir los valores para j y k . También tienden a ser muy sensitivos en la inicialización.

```
In [2]: import matplotlib.pyplot as plt
filepath = 'output.log'
with open(filepath) as fp:
    line = fp.readline()
    loggedValues = []
    uint16Values = []
    cnt = 0
    while line:
        line = line.strip().split(':')
        if (len(line)>1):
            loggedValues.append(int(line[1],16))
            #Little endian
            if (cnt%2==0):
                #LSB
                uint16Values.append(loggedValues[-1])
            else:
                #MSB
                uint16Values[-1] |= loggedValues[-1]<<8
        line = fp.readline()
        cnt+=1

    #print(uint16Values)
    arrayHistogram = plt.hist(uint16Values,bins=512)
    #print(arrayHistogram[0]) #Histogram values
    #print(arrayHistogram[1]) #Bins edges
    plt.title("Histogram of Logged Output ")
    plt.show()

    loggedBytes= bytearray(loggedValues)
    #print(loggedBytes)
    with open('output.bin','wb') as fpOutput:
        fpOutput.write(loggedBytes)
```



In []:

Figura 1. Ejecución del script *parseLog.py*



 output.bin	4/12/2020 12:11 PM	BIN File	2,040 KB
 output	4/12/2020 12:11 PM	WinRAR ZIP archive	2,016 KB

Figura 2. Comparación entre el archivo binario y el comprimido

Comentarios

En esta práctica, se exploró las instrucciones de AVR, la cual difieren un poco al ensamblador x86, específicamente en el nombre de las instrucciones y el manejo de los registros de 8-bits, lo cual se tiene que realizar otras operaciones para utilizar datos de 16-bit, lo cual lo hace un poco más revoltoso de manejar. Otra cuestión fue exportar los resultados de 1 millón de producciones, y más problema fue la parte de concatenar la parte alta y la parte baja del número de 16-bits, y más por la cantidad de producciones.

Conclusiones

Al haber concluido la práctica, se puede observar que al final de cuentas por la alta producción de números, se llega un punto donde se nota un patrón, lo cual indica que los números no son aleatorios, pero que funcionan para simular en cantidades más pequeñas. Otro factor, es el que los parámetros de la formula utilizada son constantes, solo hay un valor que cambia en cuanto n (número de muestras) va incrementando, por tanto en un punto dado se llega al patron mencionado anteriormente.

Bibliografía.

<https://www.geeksforgeeks.org/pseudo-random-number-generator-prng/>

https://en.wikipedia.org/wiki/Lagged_Fibonacci_generator

https://es.wikipedia.org/wiki/Generador_lineal_congruencial