Universidad Autónoma de Baja California

Facultad de ciencias químicas e ingeniería

Materia.

Microprocesadores y Microcontroladores.

Maestro.

Garcia Lopez Jesus Adan.

Alumno

Gonzalez Cardiel Luis Enrique

Matricula:

1217258

Grupo:

561

Trabajo:

Practica No. 6

18/10/2018

Práctica 6

• Programación en Lenguaje Ensamblador del ATmega1280

Objetivo: Mediante esta práctica el alumno aprenderá el uso básico de programa AVR Studio como herramientas de programación en lenguaje Ensamblador para el sistema ATmega1280.

Equipo: - Computadora Personal

Teoría: - Arquitectura interna y conjunto de instrucciones del ATmega1280

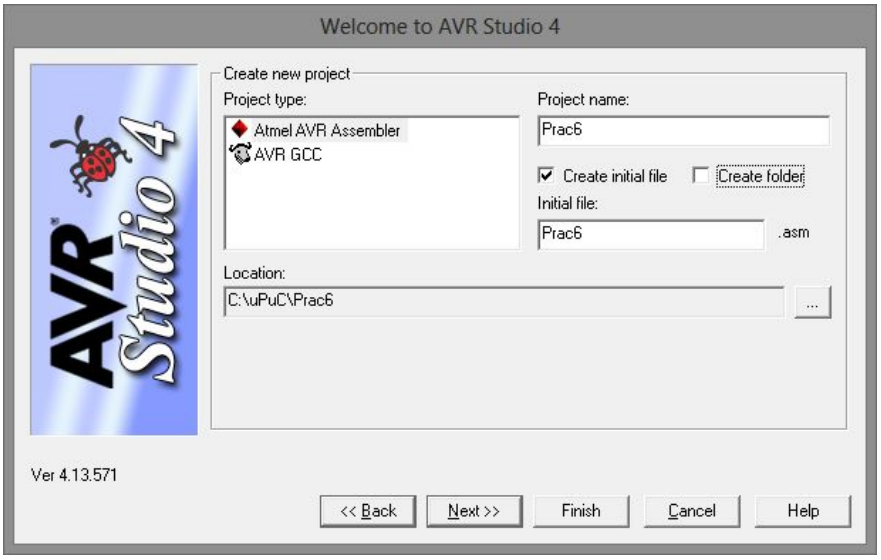
Descripción: Instalar los siguientes programas en la Computadora personal

1) AVR Studio 4 (programa IDE de la compañía ATMEL, bajar de www.atmel.com).
   Nota: actualiza mediante la instalación del service pack.
2) Crear un archivo texto con extensión asm con el código del Listado 1.

3) Crear en la raíz (C:\) una carpeta llamada uPuC y dentro de esta otra llamada Prac6 quedando la ruta como: “C:\uPuC\Prac6”.

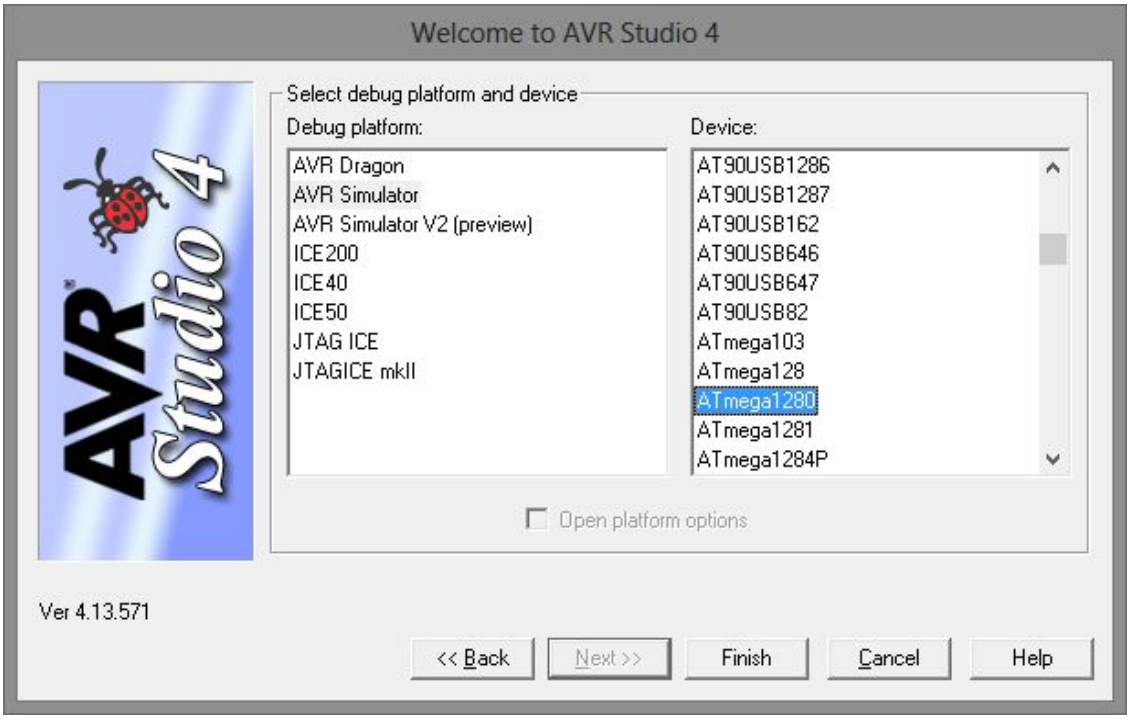
4) Utilice el programa AVR Studio para crear un proyecto llamado Prac6 llevando acabo los siguientes pasos.

a) Ejecute al programa AVR Studio y genere un proyecto seleccionando el botón New Project en la ventana de inicio para dar presentación a la siguiente ventana.

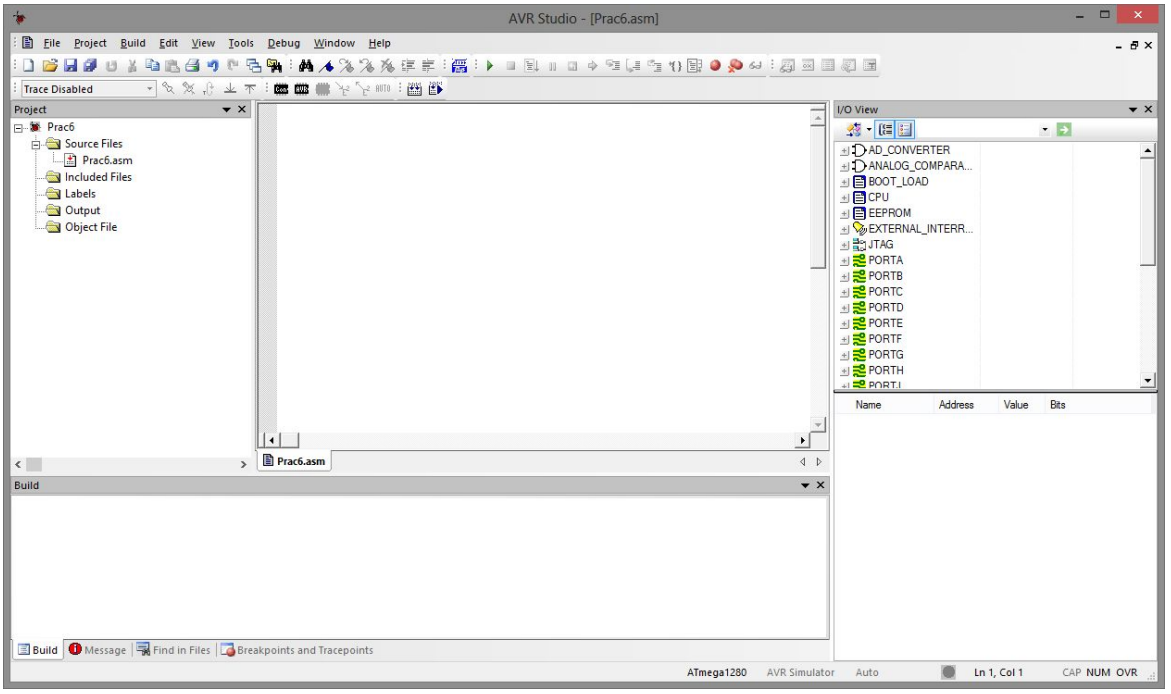


b) En esta nueva ventana seleccione Atmel AVR Assembler para indicar que utilizará el lenguaje ensamblador. Luego introduzca un nombre Prac6 al proyecto (Project name:) e introduzca la ruta c:\uPuC (Location:), desactive la opción “Create inicial file” y entonces presione el botón Next.

c) Ahora seleccione AVR Simulator como la plataforma de depuración (Debug Platform) y el dispositivo (Device) a utilizar el ATmega1280 y presione Finish .



d) Aparecerá la ventana siguiente ventana de AVR Studio para dar lugar a incluir los archivos fuentes (*.asm) al proyecto.



Listado 1:

```
;------------- definiciones e includes ------------------------------
.INCLUDE "m1280def.inc" ; Incluir definiciones de Registros para 1280
.equ INIT_VALUE  = 0      ; Valor inicial R24

;------------- inicializar ------------------------------------------
            ldi R24,INIT_VALUE

;------------- ciclo principal --------------------------------------

 arriba:    inc   R24
            cpi   R24,10
            breq  abajo
            rjmp  arriba

 abajo:     dec   R24
            cpi   R24,0
            breq  arriba
            rjmp  abajo
;--------------------------------------------------------------------
```

e) Si es necesario incluir archivos se debe posicionar el cursor en texto “Source Files” de la ventana izquierda. Presione el botón derecho del ratón y seleccione “Add Existing Source File(s)…” entonces aparecerá la ventana de selección de archivos y selecciona el(los) archivo(s) (*.asm) y se presiona Open. En este caso solo copie el código del listado 1 y salve el archivo correspondiente (Prac6.asm).

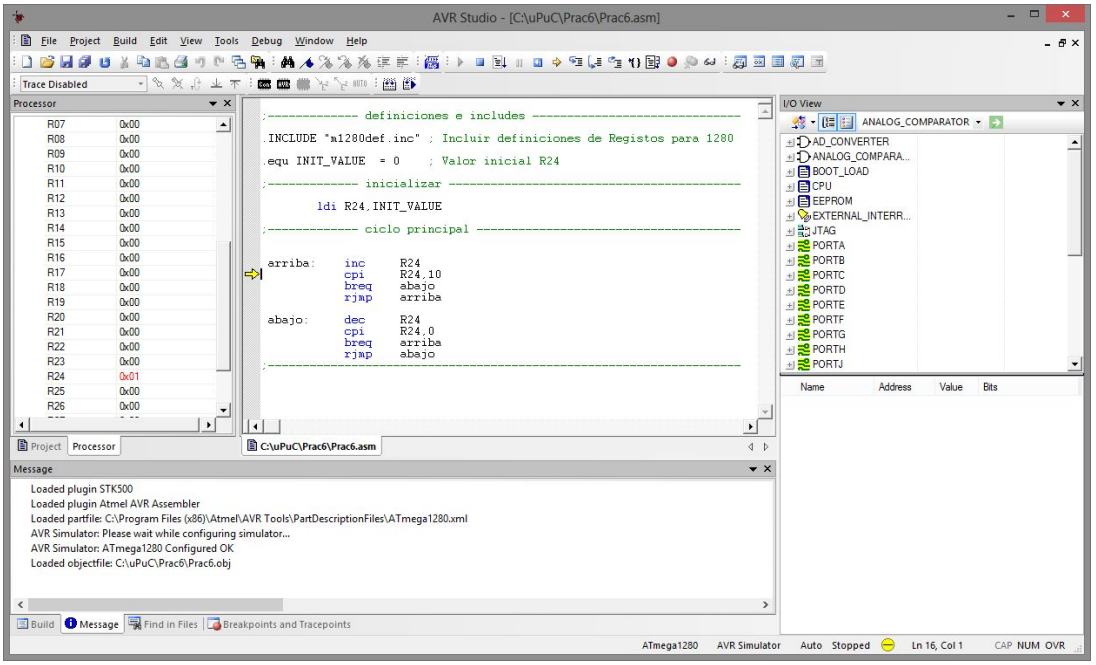
f) Ahora podrá compilar el proyecto presionando F7 o el ícono de la barra de herramientas correspondiente a compilar.

g) Una vez compilado el proyecto este puede se situado seleccionando el ícono del simulador.

h) Una vez seleccionado el simulador el programa puede ser ejecutado paso a paso o de forma automática.

i) Ensamble y simule paso a paso y observe el cambio en el registro correspondiente.

j) ¿Qué hace el programa?

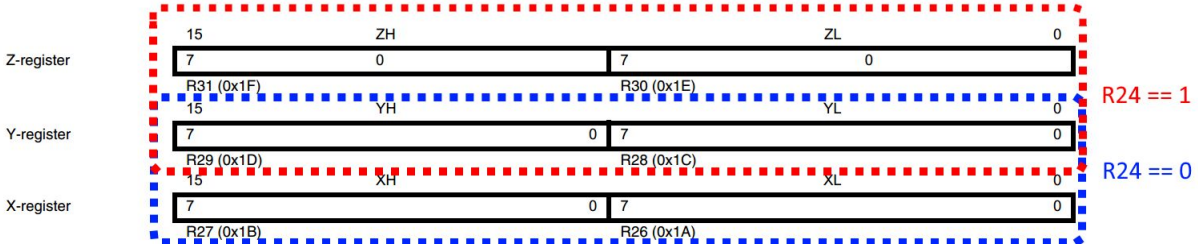


Actividades a realizar:

Escriba un programa que invierte la posición de bits, intercambiando el orden de bits del mas significativo al menos significativo, del valor dado por:

I. Los registros R31-R28 si R24 es 1,

II. Los registros R29-R26 si R24 es 0,



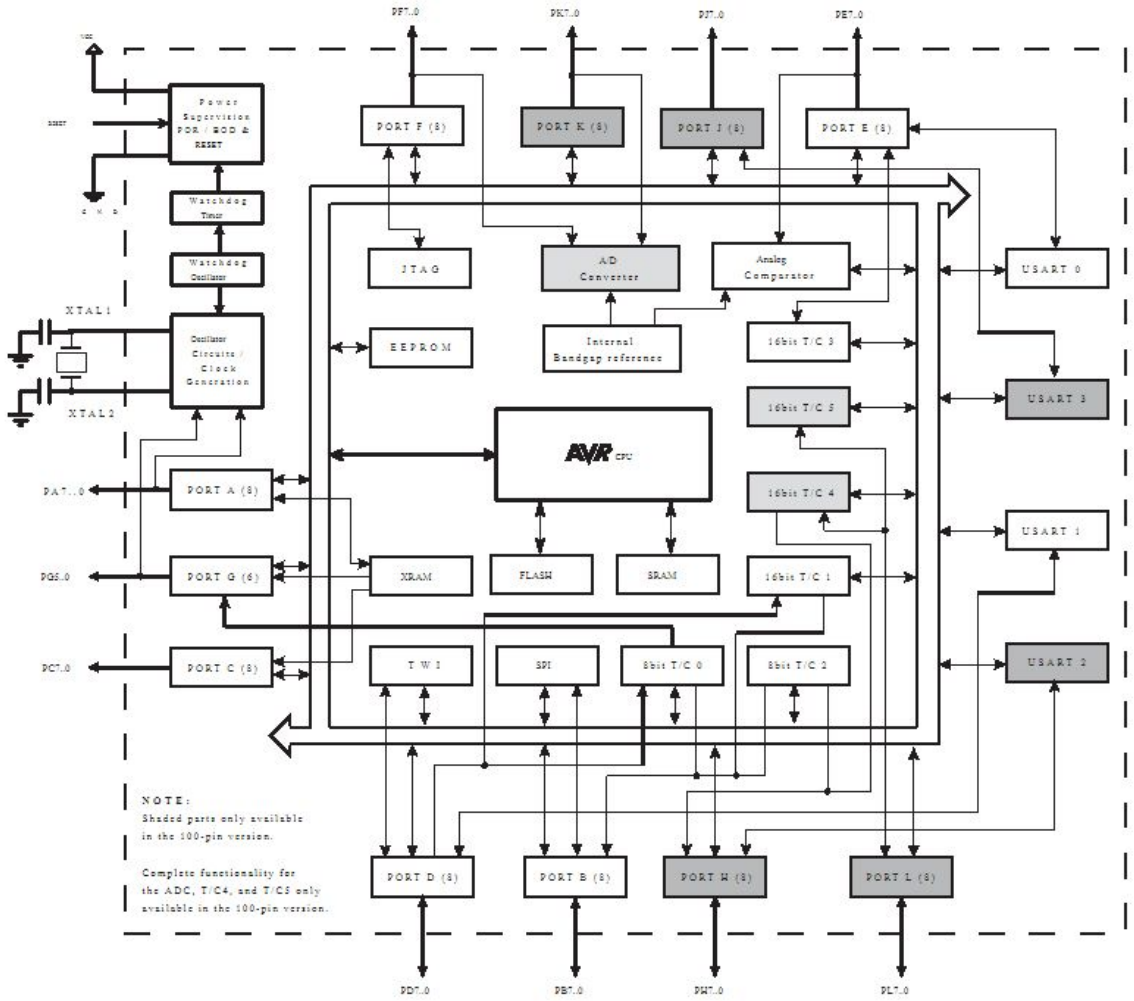
III. El programa realizará ninguna accion para cualquier otro valor en R24.

Nota: El programa debe realizar estas acciones indefinidamente.

Teoría

Features

- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 135 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 × 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16MHz
  - On-Chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 64K/128K/256KBytes of In-System Self-Programmable Flash
  - 4Kbytes EEPROM
  - 8Kbytes Internal SRAM
  - Write/Erase Cycles:10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
    - Endurance: Up to 64Kbytes Optional External Memory Space
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix acquisition
  - Up to 64 sense channels
- JTAG (IEEE® std. 1149.1 compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - Four 16-bit Timer/Counter with Separate Prescaler, Compare- and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four 8-bit PWM Channels
  - Six/Twelve PWM Channels with Programmable Resolution from 2 to 16 Bits (ATmega1281/2561, ATmega640/1280/2560)
  - Output Compare Modulator
  - 8/16-channel, 10-bit ADC (ATmega1281/2561, ATmega640/1280/2560)
  - Two/Four Programmable Serial USART (ATmega1281/2561, ATmega640/1280/2560)
  - Master/Slave SPI Serial Interface
  - Byte Oriented 2-wire Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 54/86 Programmable I/O Lines (ATmega1281/2561, ATmega640/1280/2560)
  - 64-pad QFN/MLF, 64-lead TQFP (ATmega1281/2561)
  - 100-lead TQFP, 100-ball CBGA (ATmega640/1280/2560)
  - RoHS/Fully Green
- Temperature Range:
  - -40°C to 85°C Industrial
- Ultra-Low Power Consumption
  - Active Mode: 1MHz, 1.8V: 500µA
  - Power-down Mode: 0.1µA at 1.8V
- Speed Grade:
  - ATmega640V/ATmega1280V/ATmega1281V:
    - 0 - 4MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
  - ATmega2560V/ATmega2561V:
    - 0 - 2MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
  - ATmega640/ATmega1280/ATmega1281:
    - 0 - 8MHz @ 2.7V - 5.5V, 0 - 16MHz @ 4.5V - 5.5V
  - ATmega2560/ATmega2561:
    - 0 - 16MHz @ 4.5V - 5.5V


PF7..0
PK7..0
PJ7..0
PE7..0
VCC
RESET
G N D
Power Supervision POR / BOD & RESET
Watchdog Timer
Watchdog Oscillator
XTAL1
XTAL2
Oscillator Circuits / Clock Generation
PORT F (8)
PORT K (8)
PORT J (8)
PORT E (8)
JTAG
A/D Converter
Analog Comparator
USART 0
EEPROM
Internal Bandgap reference
16bit T/C 3
16bit T/C 5
USART 3
AVR CPU
PA7..0
PORT A (8)
16bit T/C 4
USART 1
PG5..0
PORT G (6)
XRAM
FLASH
SRAM
16bit T/C 1
PC7..0
PORT C (8)
TWI
SPI
8bit T/C 0
8bit T/C 2
USART 2
NOTE:
Shaded parts only available in the 100-pin version.
Complete functionality for the ADC, T/C4, and T/C5 only available in the 100-pin version.
PORT D (8)
PORT B (8)
PORT H (8)
PORT L (8)
PD7..0
PB7..0
PH7..0
PL7..0

8. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z, C, N, V, H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z, C, N, V, H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z, C, N, V, S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z, C, N, V, H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z, C, N, V, H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z, C, N, V, H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z, C, N, V, H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z, C, N, V, S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z, N, V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z, N, V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z, N, V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z, N, V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z, N, V	1
COM	Rd	One's Complement	Rd ← 0xFF – Rd	Z, C, N, V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd	Z, C, N, V, H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z, N, V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z, N, V	1
INC	Rd	Increment	Rd ← Rd + 1	Z, N, V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z, N, V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z, N, V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z, N, V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z, C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z, C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z, C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z, C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z, C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z, C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
EIJMP		Extended Indirect Jump to (Z)	PC ←(EIND:Z)	None	2
JMP	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	4
ICALL		Indirect Call to (Z)	PC ← Z	None	4
EICALL		Extended Indirect Call to (Z)	PC ←(EIND:Z)	None	4
CALL	k	Direct Subroutine Call	PC ← k	None	5
RET		Subroutine Return	PC ← STACK	None	5
RETI		Interrupt Return	PC ← STACK	I	5
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd – Rr	Z, N, V, C, H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N, V, C, H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N, V, C, H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2

BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z, C, N, V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z, C, N, V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z, C, N, V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z, C, N, V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z, C, N, V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2

LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
ELPM		Extended Load Program Memory	R0 ← (RAMPZ:Z)	None	3
ELPM	Rd, Z	Extended Load Program Memory	Rd ← (RAMPZ:Z)	None	3
ELPM	Rd, Z+	Extended Load Program Memory	Rd ← (RAMPZ:Z), RAMPZ:Z ←RAMPZ:Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1

Mnemonics	Operands	Description	Operation	Flags	#Clocks
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

Note: EICALL and EIJMP do not exist in ATmega640/1280/1281.
ELPM does not exist in ATmega640.

¿Qué hace el programa? Este programa nos ayuda para poder comunicarnos con las diferentes familias de Atmel y asi poder manipular a nuestro antojo las diferentes operaciones de estos microprocesadores.

Conclusión

Las instrucciones de la familia Atmel 1280 y 2560 son un poco parecidas a las instrucciones del Procesador 8086 y esto me permitió realizar la practica un poco más rápido fue interesante como la materias de Algoritmos me sirvio al analizar las instrucciones que realizaba dicho programa.

Referencias

http://html.alldatasheet.com/html-pdf/174756/ATMEL/ATMEGA1280/2379/15/ATMEGA1280.html