Universidad Autónoma de Baja California
Ingeniero en computación

Organización y Arquitectura De Computadoras

Practica 11

Nombre Del Alumno: López Mercado Brayan

Matrícula: 1280838

Grupo: 551

Docente: José Isabel García Rocha

Fecha de entrega: 16 de noviembre del 2023

Objetivo

El alumno se familiarizará con el intérprete 80x86 (modo de 16 bits) sobre la plataforma T-Juino. Esto con el fin de desarrollar programas en lenguaje ensamblador para dicha plataforma.

Introducción

La tarjeta T-Juino es una plataforma de hardware basada en un PCB con un microcontrolador y un entorno de desarrollo, diseñada para el uso de microcontroladores en proyectos multidisciplinarios.

T-Juino fue diseñado para ser compatible con la plataforma libre Arduino (específicamente el Arduino Mega) y el sistema SM8088 de UABC. El hardware consiste de una placa con un microcontrolador Atmel AVR con sus puertos de entrada/Salida expuesto mediante conectores. El microcontrolador utilizado en el diseño es el Atmega1280 y para el desarrollo de software se puede utilizar entornos de desarrollo libre como Arduino IDE, Wiring o WinAVR+ programmer NotePad.

Es importante remarcar que una diferencia con la plataforma Arduino es que en T-Juino se cuenta con máquina virtual (VM88) del procesador 80x86 de 16 bits desarrollado en UABC. Esta máquina lo hace compatible con los programas ejecutables que fueron desarrollados para la plataforma SM8088 basada en el procesador clásico Intel 80C88.

Desarrollo

Para instalar el intérprete realice los siguientes pasos:

1- Ejecutar la Aplicación Xloader, colocar el puerto, seleccionar la tarjeta y cargar el intérprete.

Para Utilizar MTTTY

1- Configurar el BaudRate a 19200, paridad none, data bits 8 y stop bit 1.
2- Dar clic en conectar.
3- Presionar cualquier tecla al aparecer el símbolo?
4- Presionar I y enter para cargar el archivo .COM o .BIN a cargar.
5- Presionar g para iniciar el programa.
6- Para terminar el programa presionar el push-Button de la tarjeta.

Basándose en el listado 1, crear un archivo texto llamado p11.asm

```
.model tiny

;----- Insert INCLUDE "filename" directives here
;----- Insert EQU and = equates here

locals

.data
      Mens  DB  'Hola Mundo',10,13,0

.code
      org 100h

;*****************************************************************
;  Procedimiento principal
;*****************************************************************
  principal    proc
               mov  sp,0fffh          ; inicializar SP (Stack Pointer)
       @@ini0: mov  dx,1
       @@ini1: mov  cx,dx
      @@sigue: mov  al,'x'
               call putchar
               loop @@sigue
               mov  al,10
               call putchar
               mov  al,13
               call putchar
               inc  dx
               cmp  dx,20
               jbe  @@ini1
               jmp  @@ini0
               ret                    ; nunca se llega aquí
               endp

;*****************************************************************
;  procedimientos
;*****************************************************************
    putchar    proc
               push ax
               push dx
               mov dl,al
               mov ah,2       ; imprimir caracter DL
               int 21h        ; usando servicio 2 (ah=2)
               pop dx         ; del int 21h
               pop ax
               ret
               endp

end   principal               ; fin de programa (file)
```

Figura 1: Código del Listado 1.

1- Instalar GUI Turbo Assembler v3.0.1 versión 64 bits si es necesario.
2- Ejecutar el programa, abrir la sección de preferencias y modificar los parámetros del linker, eliminar todos y agregar /t.
3- Cargar el archivo .asm
4- Presionar F5 para ensamblar y enlazar el archivo, se genera el archivo .obj y posteriormente el archivo .COM.
5- Cargar este último archivo con ‘l’ en la placa y ejecutarlo con ‘g’.

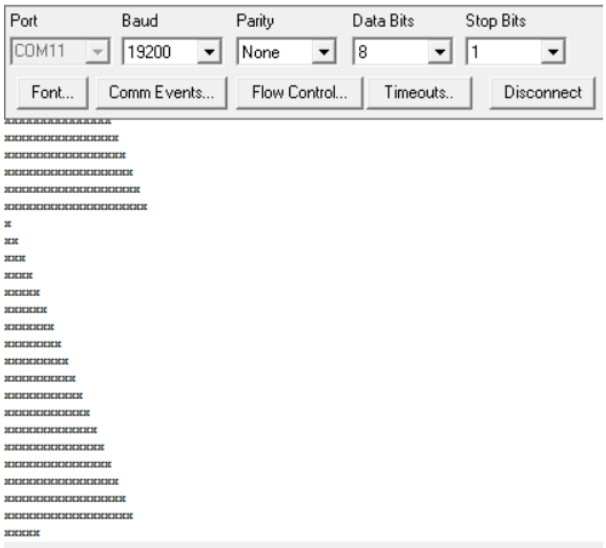

Figura 2: Código del listado 1 corriendo en la tarjeta.

A) Elaborar un diagrama de flujo detallando el proceso para realizar todas las actividades anteriormente detalladas presentadas, paso a paso.

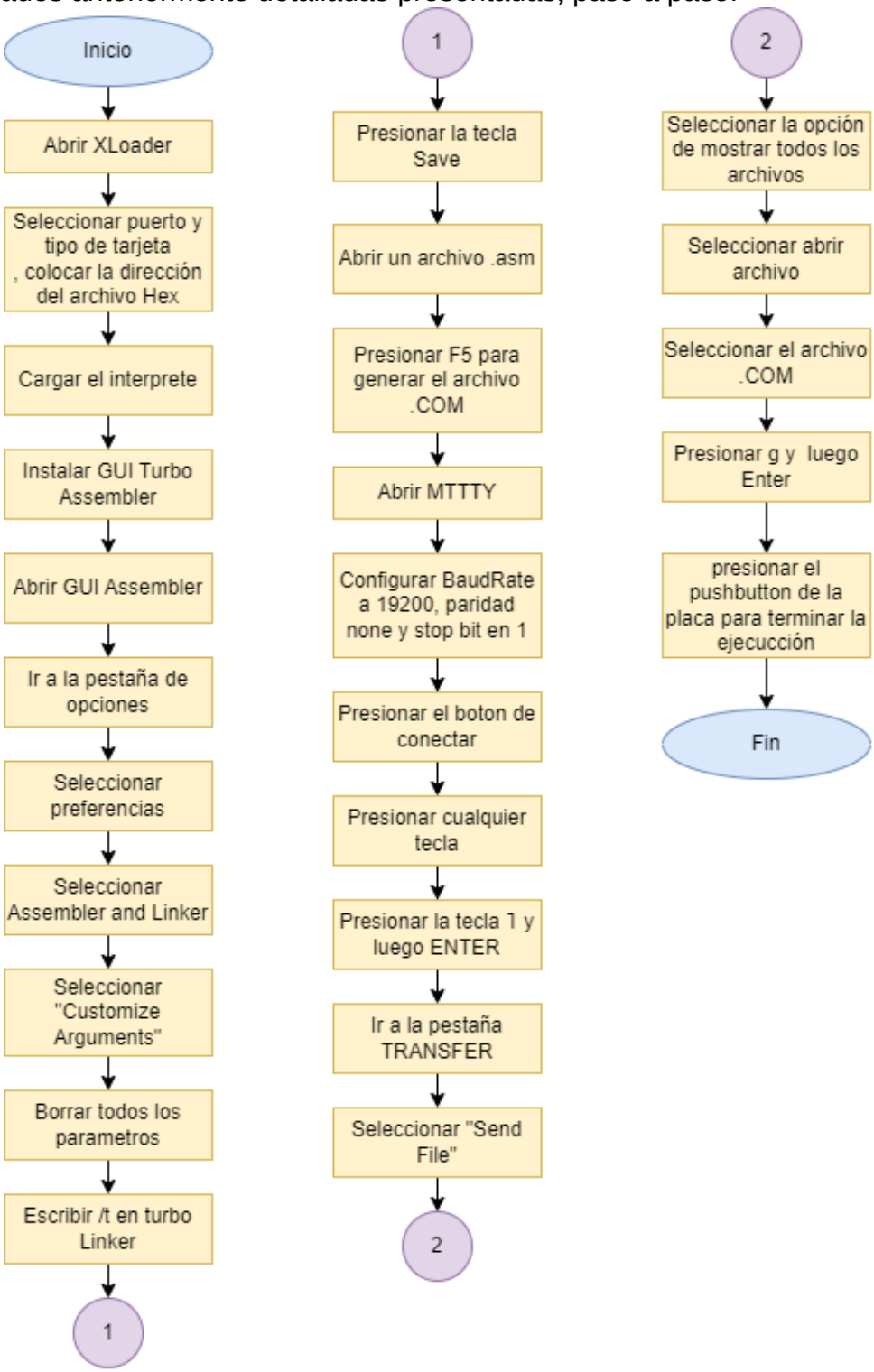

Figura 3: Diagrama De Flujo de como cargar un archivo en T-Juino por primera vez.

B) Implementar las subrutinas printDec y printHex en ensamblador 8086 y correrlo en la tarjeta.

Subrutina printDec

```
printDec proc
push ax
push bx
push cx
push dx
xor cx,cx
xor dx,dx
mov bx,10
divide:
    div bx
    push dx ;Enviar el residuo al stack
    inc cx
    xor dx,dx
    cmp ax,0 ;El numero ya es 0?
    je print ;Si
    jmp divide ;No
print:
    pop dx ;Tomar el digito del stack
    add dx,'0'
    mov ah,2h
    int 21h
    loop print
    pop dx
    pop cx
    pop bx
    pop ax
    ret
endp
```

Figura 4: Subrutina printDec versión de 16 bits.

Descripción: primero se realizan la operación con los registros CX y DX para ponerlos en cero, de manera que no afecten el resultado, el registro CX se utiliza como contador, específicamente la cantidad de dígitos que tendrá el numero decimal a mostrar y DX es el registro que contiene el residuo de las divisiones, BX se utiliza únicamente para almacenar el numero 10 decimal; dentro de la etiqueta Divide se realizan las divisiones entre AX (el número que se desea mostrar en pantalla) y BX cuyo residuo será enviado a la pila para almacenarlo de manera temporal, esto se realizó de esta manera ya que esta versión de ensamblador no permite guardar en variables (puede que si se pueda, pero se desconoce cómo hacerlo), regresando al código, una vez enviado el residuo a la pila se realiza la operación XOR DX, DX para ponerlo en 0, en caso de que el cociente (el registro AX) aun no sea 0, se realizara un salto a la etiqueta divide para repetir el proceso, en caso de que el cociente sea 0, se procederá a sacar de la pila el residuo y se luego se le sumara ‘0’ para convertirlo en carácter y con ello mostrarlo por medio del servicio 2 y la interrupción 21h, este ciclo continuara hasta mostrar todos los dígitos del número.

Subrutina printHex

```
printHex proc
 push ax
 push bx
 push cx
 push dx
 xor cx,cx
 xor dx,dx
 mov bx,10h
 .divide:
      div bx
      push dx ;Enviar el residuo al stack
      inc cx
      xor dx,dx
      cmp ax,0 ;El numero ya es 0?
      je .print ;Si
      jmp .divide ;No
.print:
      pop dx ;Tomar el digito del stack
      cmp dx,9 ;Se necesita Ajuste?
      jbe .menor ;No
      add dx,7 ;Si
      .menor:
      add dx,'0'
      mov ah,2h
      int 21h
      loop .print
      pop dx
      pop cx
      pop bx
      pop ax
      ret
endp
```

Figura 5: Subrutina printHex versión de 16 bits.

Descripción: Como se puede observar en la figura 5, la subrutina printHex tiene el mismo funcionamiento que la subrutina printDec de la figura 4, con la diferencia de que ahora en vez de dividir en un 10 decimal ahora divide entre un 10 hexadecimal, y que en ocasiones es necesario hacer un ajuste al carácter a mostrar en pantalla, es decir, debido a que los caracteres que representan a las letras A, B, C, D, E y F tienen una separación de 7 espacios respecto a los que representan a los números del 0 al 9 en la tabla ASCII, se le debe sumar un 7 al residuo en caso de que este sea mayor que 9, por medio de ese ajuste se puede mostrar números hexadecimales que contengan las letras desde A hasta F de manera correcta, en caso de que el residuo sea menor o igual a 9, no se necesita ningún ajuste y se puede convertir a carácter de manera directa.

Pruebas

Formato de salida: Numero en Hexadecimal, “espacio”, Numero en decimal.

AX= 0xFFFF

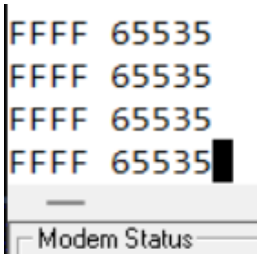

Figura 6: Prueba 1 de printHex y printDec

AX= 0xA2E

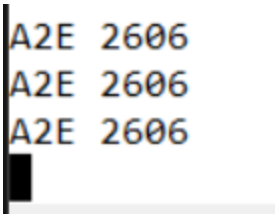

Figura 7: Prueba 2 de printHex y printDec

AX= 120d

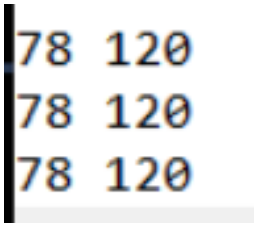

Figura 8: Prueba 3 de printHex y printDec

AX= 101011110001b

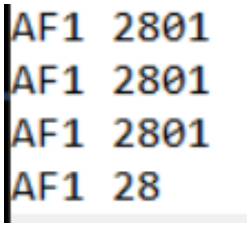

Figura 9: Prueba 4 de printHex y printDec

Código Completo

```
.model tiny
locals
.data
    NoteOfTheDayP11 DB 'Necesito Dormir',10,13,0
    NL DB 13,0
    outputDecTag DB 'Salida Con PrintDec: ',10,13,0
    outputHexTag DB 'Salida Con PrintHex',10,13,0

.code
   org 100h

Main proc
    start:mov sp,0fffh
    mov ax,0Fh
    call printHex
    mov cl,32
    call putchar
    call printDec
    call newLine
    jmp start
    ret
    endp

printHex proc
 push ax
 push bx
 push cx
 push dx
 xor cx,cx
 xor dx,dx
 mov bx,10h
 .divide:
     div bx
     push dx ;Enviar el residuo al stack
     inc cx
     xor dx,dx
     cmp ax,0 ;El numero ya es 0?
     je .print ;Si
     jmp .divide ;No
.print:
     pop dx ;Tomar el digito del stack
     cmp dx,9 ;Se necesita Ajuste?
     jbe .menor ;No
     add dx,7 ;Si
     .menor:
```

```
        add dx,'0'
        mov ah,2h
        int 21h
        loop .print
        pop dx
        pop cx
        pop bx
        pop ax
        ret
endp

 printDec proc
 push ax
 push bx
 push cx
 push dx
 xor cx,cx
 xor dx,dx
 mov bx,10
 divide:
        div bx
        push dx ;Enviar el residuo al stack
        inc cx
        xor dx,dx
        cmp ax,0 ;El numero ya es 0?
        je print ;Si
        jmp divide ;No
 print:
        pop dx ;Tomar el digito del stack
        add dx,'0'
        mov ah,2h
        int 21h
        loop print
        pop dx
        pop cx
        pop bx
        pop ax
        ret
 endp

putchar proc
 push ax
 push cx
 push dx
 mov dl,cl
 mov ah,2
```

```
 int 21h
 pop dx
 pop cx
 pop ax
 ret
 endp

newLine proc
 push cx
 mov cl,10
 call putchar
 mov cl,13
 call putchar
 pop cx
 ret
endp

end Main
```

Link de GitHub con código completo

https://github.com/BrayanLMercado/OAC_Practica11_v2.git

Conclusiones y comentarios

- El programar con lenguaje ensamblador de 16 bits a diferencia de hacerlo con el de 32 bits es que al no existir una manera de reservar espacio en memoria se debe recurrir al uso del Stack para almacenar datos de manera temporal, si esto no se realiza de manera correcta se puede perder control de cuantas veces se debe de utilizar las instrucciones push y pop.

- La manera de mostrar caracteres cambia respecto al de 32 bits, es decir, en el de 16, DX recibe el dato a mostrar en lugar de CX y el servicio para mostrar en consola ahora se realiza mandando un 2 en AH, en lugar de AX y la interrupción a usar es la 21h en lugar de la 80h.

- En caso de que desee visualizar el código con colores, le recomiendo que use el link de GitHub que se anexo a la práctica, así se puede evitar daños en los ojos a causa del fondo blanco del documento (Sí, esta es la tercera vez que pongo este comentario).

Dificultades en el desarrollo

- La primera dificultad real fue el tener que aprender a utilizar el Stack para guardar datos de manera temporal y mostrarlos en el orden correcto, aunque esto último no dio problemas en realidad por el tipo de pila que maneja el ensamblador.

- Debido a que en el semestre anterior tuve que instalar el GUI Turbo Assembler, me ahorre tiempo en instalarlo y configurarlo, aunque tuve que volver a

acostumbrarme a usar un IDE con fondo blanco (aunque luego encontré como cambiarle los colores).

- Debido a que este ensamblador se ejecuta de manera indefinida, se tuvo que buscar una manera de separar las líneas para que fuera fácil leerlas y distinguirlas, al final se hizo la subrutina newLine para ello.

- El intentar adaptar la subrutina printHex que se nos dio en la practica 5 fue un dolor de cabeza debido a que no me resultaba sencillo el saber dónde colocar la etiqueta para el código que imprimiera los caracteres, la forma fácil de evitar este problema fue tomar la subrutina printDec y hacerle modificaciones para que imprimiera en Hexadecimal, y solo fue necesario agregarle líneas extras para el ajuste de los caracteres, puede que no sea tan optima como la original, pero fue más sencilla de implementar.

Referencias.