UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Estructura y Programación de Computadoras.

Grupo: 04

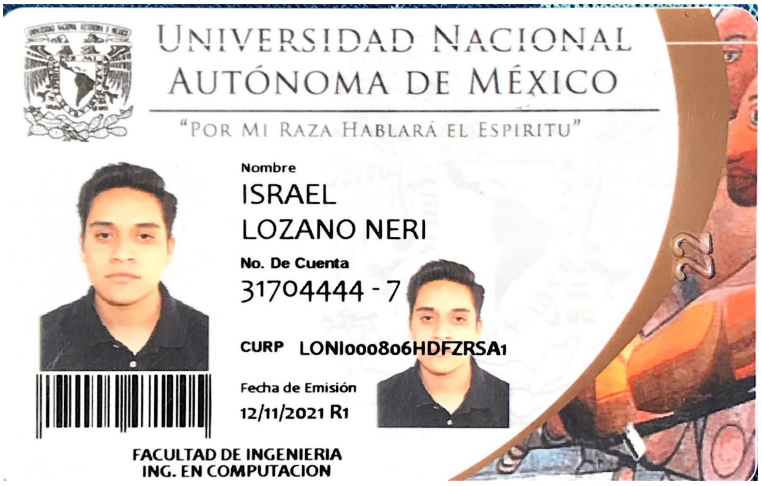
PROYECTO 1
“Compilador básico para el MC68HC11 de Motorola”

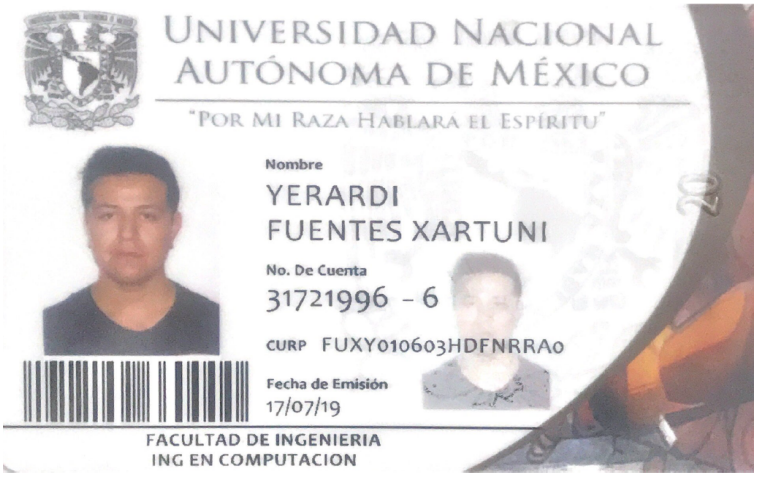
Profesor: Pedro Ignacio Rincón Gómez

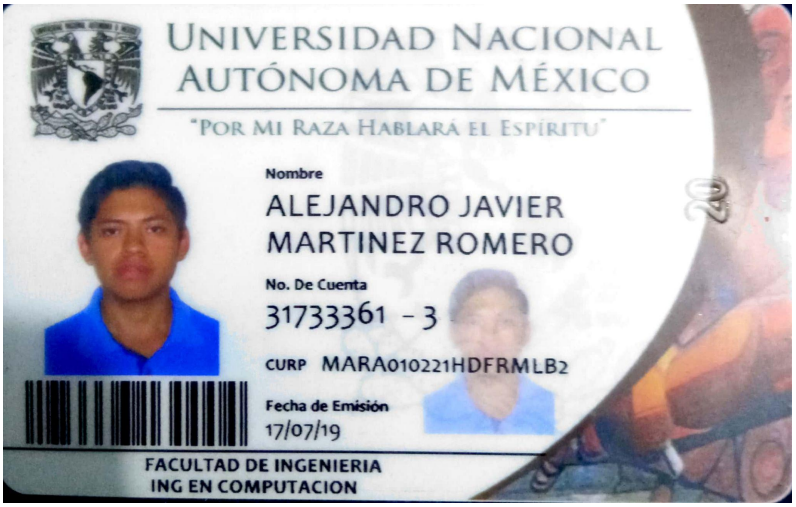
Integrantes:

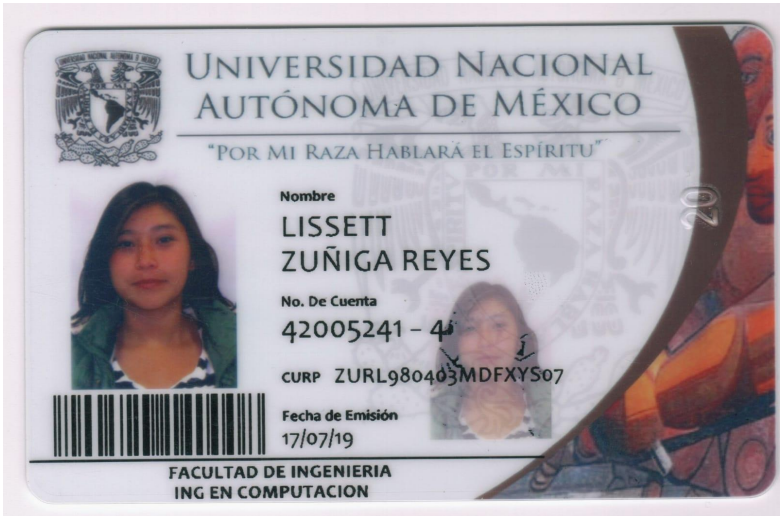
- **Fuentes Xartuni Yerardi**
- **Lozano Neri Israel**
- **Martinez Romero Alejandro Javier**
- **Zuñiga Reyes Lissett**

Copia legible de una identificación con fotografía en formato digital.






UNIVERSIDAD NACIONAL
AUTÓNOMA DE MÉXICO
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Nombre
ALEJANDRO JAVIER
MARTINEZ ROMERO
No. De Cuenta
31733361 - 3
CURP MARA010221HDFRMLB2
Fecha de Emisión
17/07/19
FACULTAD DE INGENIERIA
ING EN COMPUTACION


UNIVERSIDAD NACIONAL
AUTÓNOMA DE MÉXICO
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Nombre
LISSETT
ZUÑIGA REYES
No. De Cuenta
42005241 - 4
CURP ZURL980403MDFXYS07
Fecha de Emisión
17/07/19
FACULTAD DE INGENIERIA
ING EN COMPUTACION

Criterios de diseño y estructura que guarda el compilador.

Programa Principal:

El código crea una lista con todos los mnemónicos que obtiene del archivo de excel, también, crea una lista con las directivas de ensamblador las cuales son, ORG,EQU.
Posteriormente, el programa verifica que el archivo a leer esté en formato ASC, en caso de que el archivo sea aceptado, se abre el archivo y comienza a leer cada uno de los párrafos.
Para los comentarios indicamos que cada que se identifique un asterisco (*) se separará la cadena con la función .split a partir de dicho carácter las cadenas que no son comentarios se quedan y las almacenamos en una nueva variable para que una vez que tengamos el programa sin comentarios, podamos continuar trabajando con él.
Después, buscamos las directivas ORG y END que fueron previamente almacenadas en una variable ya que sin estas no se podrá compilar el archivo, en caso de que no existan estas palabras, el programa arroja un mensaje indicando que no se encuentran dichas directivas.

Para la lectura y reconocimiento de los tipos de direccionamiento se clasificara por etiquetas y variables guardandolas en otros glosarios para así poder comenzar a clasificar las instrucciones, para las etiquetas se guardará la posición de memoria donde se localiza el tipo de direccionamiento, por otro lado, cuando se lee una línea que no contiene instrucción alguna la marcaremos como línea “vacía”.
Una vez realizado este proceso, comienza con la lectura de mnemónicos verificando que ya hayan sido identificados, procede a reconocer los modos de direccionamiento que tiene, en caso de que no hayan sido identificados exitosamente se manda una advertencia al usuario.

En el código se emplearon sentencias if-else para identificar los diferentes tipos de direccionamiento, dependiendo cual sea el que se esté leyendo se accede al diccionario de mnemónicos en donde se obtendrá el número de bytes correspondientes a cada operación para posteriormente hacer la interpretación.
Al reconocer el carácter “#” sabemos que el tipo de direccionamiento es inmediato, si se reconoce una “x” o “y” entonces el tipo de direccionamiento es indexado, en caso de que sea el direccionamiento relativo, se guarda la dirección de memoria en un nuevo directorio donde se encontró, sí se encuentra una directiva, etiqueta o demás, se omite y sigue leyendo.

Para el modo de direccionamiento inherente, se verifica que no contenga ningún operando. Si no es inmediato, se verifica el operando para convertirlo a ASCII o hexadecimal, dependiendo el caso puede dejarse tal y como está o cambiar el valor por el de la variable

(en caso de que sea una variable) o una etiqueta, se deja como etiqueta. Para el direccionamiento indexado, se quita del operando x o y.

El programa genera un archivo de texto con extensión LST el cual contiene el número de línea, código objeto y el código fuente, este formato lo pudimos obtener con el método llamado contador, el cual recibe como parámetros la dirección de memoria, el código objeto, el mnemónico, etc.

Finalmente, se genera un archivo de texto con extensión S19 que contiene al código objeto con el formato solicitado, estos puntos se pueden observar claramente al final de nuestro código, es decir, cuando nuestra sentencia Error es igual a falso, significa que el proceso de lectura del archivo fue correcto y que la extensión fue la apropiada, entonces generará los dos archivos y escribirá sobre ellos mediante la función write(), de esta manera nosotros podremos visualizar el contenido de ambos archivos generados.

Métodos:

- **def buscar(texto):** Este método se encarga de buscar en el archivo la palabra ORG de modo que si esta se encuentra nos regresa la dirección de memoria, de lo contrario se arroja un mensaje indicando que es falso.
- **def noHexadec(operando):** Método usado para comprobar que se solo se contengan caracteres en sistema hexadecimal.
- **def cronometro(diccionario, cifras, ciclos, datos, var):** Realiza la suma de bytes necesaria a la dirección de memoria de mnemónicos.
- **def contador(dirMemoria, ciclos, codigoObjeto, mnem, op, pos):** Se usa para la obtención del código objeto y para conocer la dirección de memoria.
- **def cargar():** Con ayuda de este método vamos a abrir nuestro archivo "csv" el cual tiene todos los mnemónicos,mismos que serán guardados para después cargarlos.
- **def hexadec(val, nbits):** Con este método simplemente vamos a hacer una conversión de números al sistema hexadecimal.

Evidencias de cada uno de los puntos que sustentan la calificación (Archivos, capturas de pantalla, etc.):

El compilador programado deberá reconocer todos los mnemónicos del set de instrucciones del MC68HC11 tanto en letras mayúsculas como minúsculas, así como la sintaxis correspondiente a cada uno de los seis modos de direccionamiento que soporta el CPU de dicho microcontrolador.

Direccionamiento inmediato:

```
if("#" in linea):
    #(direccionamiento inmediato)
    if mnemonicos[mnemo][0][0] != "-- ":
        codigoFuente = codigoFuente + "  " + str(i+1) + ": " + dirMemoria + "(" + (mnemonicos[mnemo][0][0]).replace(" ", "") + operando.replace(" ", "") + "):" + mnemo + "
        dirMemoria, codigoObjeto, pos = contador(dirMemoria, int(mnemonicos[mnemo][0][2]), codigoObjeto, mnemonicos[mnemo][0][0], operando, pos)
        if i in comentario:
            codigoFuente = codigoFuente + " " + comentario[i] + "\n"
        else:
            codigoFuente = codigoFuente + "\n"
    else:
        print("No hay modo de direccionamiento linea: ", i+1)
elif(",X" in linea):
    if len(operando.replace(",X", "")) > 3:
        print("MAGNITUD DE OPERANDO ERRONEA EN LINEA: ", i+1)
        Error = True
```

Direccionamiento indexado en X:

```
    #(direccionamiento indexado en x)
    elif mnemonicos[mnemo][2][0] != "-- ":
        codigoFuente = codigoFuente + "  " + str(i+1) + ": " + dirMemoria + "(" + mnemonicos[mnemo][2][0].replace(" ", "") + operando.replace(" ", "").replace(",X", "") + "
        operando = operando.replace(",X", "")
        dirMemoria, codigoObjeto, pos = contador(dirMemoria, int(mnemonicos[mnemo][2][2]), codigoObjeto,mnemonicos[mnemo][2][0], operando, pos)
        if i in comentario:
            codigoFuente = codigoFuente + " " + comentario[i] + "\n"
        else:
            codigoFuente = codigoFuente + "\n"
    else:
        print("No hay modo de direccionamiento linea: ", i+1)
elif(",Y" in linea):
    if len(operando.replace(",Y", "")) > 3:
        print("MAGNITUD DE OPERANDO ERRONEA EN LINEA: ", i+1)
        Error = True
```

Direccionamiento indexado en Y:

```
    #(direccionamiento indexado en y)
    elif mnemonicos[mnemo][3][0] != "-- ":
        codigoFuente = codigoFuente + "  " + str(i+1) + ": " + dirMemoria + "(" + (mnemonicos[mnemo][3][0]).replace(" ", "") + operando.replace(" ", "").replace(",Y", "") +
        operando = operando.replace(",Y", "")
        dirMemoria, codigoObjeto, pos = contador(dirMemoria, int(mnemonicos[mnemo][3][2]), codigoObjeto,mnemonicos[mnemo][3][0], operando, pos)
        if i in comentario:
            codigoFuente = codigoFuente + " " + comentario[i] + "\n"
        else:
            codigoFuente = codigoFuente + "\n"
    else:
        print("No hay modo de direccionamiento linea: ", i+1)
```

Direccionamiento directo:

```
elif(len(operando)<3):
    #(direccionamiento directo)
    if mnemonicos[mnemo][1][0] != "-- ":
        codigoFuente = codigoFuente + "  " + str(i+1) + ":" + dirMemoria + "(" + (mnemonicos[mnemo][1][0]).replace(" ", "") + operando.replace(" ", "") + "):" + mnemo + " "
        dirMemoria, codigoObjeto, pos = contador(dirMemoria, int(mnemonicos[mnemo][1][2]), codigoObjeto, mnemonicos[mnemo][1][0], operando, pos)

        if i in comentario:
            codigoFuente = codigoFuente + " " + comentario[i] + "\n"
        else:
            codigoFuente = codigoFuente + "\n"
    else:
        print("No hay modo de direccionamiento linea: ", i+1)
elif mnemonicos[mnemo][6][0] != "-- ":
    codigoFuente = codigoFuente + "  " + str(i+1) + ": " + dirMemoria + "(" + (mnemonicos[mnemo][6][0]) + " " + operando + "):" + mnemo + " " + lineas[1]
    dirMemoria, codigoObjeto, pos = contador(dirMemoria, int(mnemonicos[mnemo][6][2]), codigoObjeto, mnemonicos[mnemo][6][0], "RL", pos)
    direccionamientoRelativo[operando.replace(" ","")] = dirMemoria
    if i in comentario:
        codigoFuente = codigoFuente + " " + comentario[i] + "\n"
    else:
```

Direccionamiento extendido:

```
        #(Direccionamiento extendido)
        if mnemonicos[mnemo][4][0] != "-- ":
            codigoFuente = codigoFuente + "  " + str(i+1) + ": " + dirMemoria + "(" + (mnemonicos[mnemo][4][0]).replace(" ", "") + operando.replace(" ", "") + "):" + mne
            dirMemoria, codigoObjeto, pos = contador(dirMemoria, int(mnemonicos[mnemo][4][2]), codigoObjeto,mnemonicos[mnemo][4][0], operando, pos)
            if i in comentario:
                codigoFuente = codigoFuente + " " + comentario[i] + "\n"
            else:
                codigoFuente = codigoFuente + "\n"
        else:
            print("No hay modo de direccionamiento linea: ", i+1)
if mnemo in palabrasClave:
  continue
if mnemo in etiquetas:
  continue
se:
  print("NO EXISTE EL MNEMONICO ", mnemo, "En linea: ", i)
  Error = True
```

Direccionamiento relativo:

```
#(direccionamientoRelativo)

if mnemo in mnemonicos:
    #print(mnemo, operando)
    if mnemonicos[mnemo][6][0] != "-- " and operando not in etiquetas:
        print("ETIQUETA INEXISTENTE EN LINEA: ", i+1)
        Error = True
        continue

if len(lineas) > 1:
    operando = lineas[1]
    #print("LINEAS 1", lineas[1])

if operando in direccionamientoRelativo and len(lineas) > 1:
    #(direccionamientoRelativo[operando], etiquetas[operando])
    if int(direccionamientoRelativo[operando],16) > int(etiquetas[operando],16):
        if str(hex(int(etiquetas[operando],16) - int(direccionamientoRelativo[operando],16))[3:]).isdecimal():
            resHex = -int(hex(int(etiquetas[operando],16) - int(direccionamientoRelativo[operando],16))[3:])
        else:
            resHex = -int(hex(int(etiquetas[operando],16) - int(direccionamientoRelativo[operando],16))[3:],16)
        #print("LEN DE LINEAS:", len(lineas))
        #print(lineas)
        if int(resHex) < -127:
            print("SALTO RELATIVO MUY LEJANO EN LINEA: ", g+1)
```

El compilador deberá soportar archivos documentados con comentarios, los cuales tendrá que ignorar.

```
************************************
* VECTOR INTERRUPCION SERIAL
************************************
        ORG    $FFD6
        FCB    $F1,$00


************************************
*RESET
************************************
```

El compilador deberá reconocer las directivas de ensamblador “ORG”, “EQU”, ”FCB” y “END”.

Se añade una captura donde se pueden ver algunas de las directivas de ensamblador las cuales se encuentran en el código de prueba.

CodigoPrueba.asc: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
COLUMNA EQU $45
MEMORIA EQU $1789

ORG  $8000

PACTL  EQU   $1026
PACNT  EQU   $1027
ADCTL  EQU   $1030
ADR1   EQU   $1031
ADR2   EQU   $1032
ADR3   EQU   $1033
ADR4   EQU   $1034
OPTION EQU   $1039
```

El compilador será capaz de abrir un archivo de texto codificado en ANSI, con extensión "*.asc", que contenga el código fuente en lenguaje ensamblador. Después procederá a hacer un análisis del mismo, línea por línea (hasta llegar al END), para determinar el código objeto correspondiente.

Se muestra el mensaje de salida del programa al codificar el archivo prueba.

```
Se ha compilado el archivo de manera correcta *_*

In [2]:
```

El compilador generará un archivo de texto con extensión "*.LST" que contenga el código fuente y el código objeto correspondiente.
El compilador generará un archivo de texto con extensión "*.S19"

CodigoPrueba.asc	906	334	Archivo ASC	07/05/2022 08:...	35EFEA41
CodigoPrueba.LST	1,699	615	MASM Listing	07/05/2022 09:...	0FA2934A
CodigoPrueba.S19	190	127	Archivo S19	07/05/2022 09:...	B96347E0

Archivo con extensión ".LST":

```
1: VACIO COLUMNA EQU $45
2: VACIO MEMORIA EQU $1789
3: VACIO
4: VACIO ORG  $8000
5: VACIO
6: VACIO PACTL  EQU   $1026
7: VACIO PACNT  EQU   $1027
8: VACIO ADCTL  EQU   $1030
9: VACIO ADR1   EQU   $1031
10: VACIO ADR2   EQU   $1032
11: VACIO ADR3   EQU   $1033
12: VACIO ADR4   EQU   $1034
13: VACIO OPTION EQU   $1039
14: VACIO
15: VACIO PORTA  EQU   $1000
16: VACIO PORTD  EQU   $1008
17: VACIO PORTE  EQU   $100A
18: VACIO PORTG  EQU   $1002
19: VACIO
```

Archivo con extensión ".S19":

CodigoPrueba: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
<8000> 18 09 3D 8F 18 8F C6 0c 86 48 CC 23 45 CE 05 fd
<8010> 8B 4C 84 F0 18 CE HO LA A6 29 18 E6 34 CD EE F1
<8020> 18 AB 23 B4 F4 18 EE AB B6 03 4D F6 00 7b FE fd
<8030> e8 BB 07 CB
```