UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

Asignatura: Estructura y Programación de Computadoras (1503)
Proyecto 1: Compilador básico del MC68HC11

Profesor: M.I Pedro Ignacio Rincón Gómez

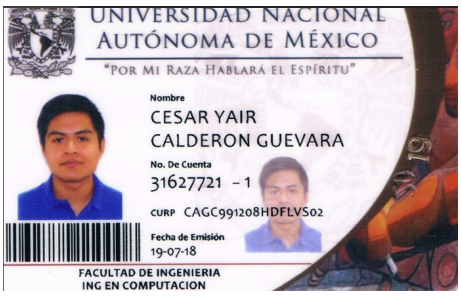
Integrantes del equipo:

- *Bautista Pérez Brian Jassiel*
- *Calderón Guevara César Yair*
- *Maceda Patricio Fernando*
- *Vázquez Flores José Martín*

Fecha de entrega: 27 de julio de 2021

1. Identificaciones de los integrantes

(a) Bautista Pérez Brian Jassiel



(b) Calderón Guevara César Yair

(a) Maceda Patricio Fernando

(b) Vázquez Flores José Martín

Integrantes del equipo que realizó el proyecto.

2. Descripción del proyecto y aportaciones

2.1 Módulo científico pandas e importaciones

Tenemos la implemación del módulo **pandas** el cual nos servirá para leer el archivo de Excel en donde están los comandos del MC68HC11.

```
1 import pandas as pd              # Biblioteca que nos permitirá leer el Excel con codigo de
      instrucciones
2 from string import hexdigits     # Para poder escribir valores .s19 en el archivo
```

2.2 Funciones

En un principio tenemos la creación de 16 funciones (implementadas por el integrante César Yair Calderón Guevara), las cuales en términos generales va a definir las operaciones del compilador del MC68HC11, a continuación daremos una breve explicación de lo que hace cada función.

- INICIO : Va a identificar la palabra reservada ORG y se marcará el inicio del programa.
- fun_cambio_formato: Cambiará de formato decimal a hexadecimal.
- fun_clr_valor: Quita caracteres para el LST.
- fun_verifica_long: Verifica que las variables y las palabras reservadas estén bien escritas.

- fun_verifica_etiqueta: Verifica que las etiquetas estén escrita y añade sus valores.
- fun_salto_relativo: Verifica que el salto relativo no exceda los valores de memoria.
- ajuste_De_linea: Formatea las cadenas para que se ajusten los colores de impresión
- quita_comentarios:Quita las cadenas de palabras para generar el archivo que no contenga estos comentarios.
- BNE : Verifica que el valor sea igual a cero para redireccionar a memoria.
- BRCLR : Si los bits valen cero, conduce a una operación.
- EQU : Asigna la etiqueta de valor dado para que pueda ser asignado.
- END : Marcamos el fin del programa.
- FCB : Genera un byte constante para agregar datos.
- JMP : Realizar los saltos en memoria a etiquetas dadas.
- NOP : No realiza ninguna operación, lanza un error en caso de encontrar un valor.
- LDX : Cargar dato de 16 bits en el indice de X y en el indice de Y.

```
def INICIO(inicia_memoria_org,tag,op_name):     # Llamado ORG, define el inicio del programa
    if not programa.comienzo:
        programa.inicio_memoria = fun_clr_valor(inicia_memoria_org)      #inicio programa Memoria en
             base hexadecimal
        programa.posicion_memoria = int(programa.inicio_memoria,16) #Nos devuelve al valor del
            inicio del programa en entero
        programa.comienzo = True
    fun_verifica_long(programa.inicio_memoria, 2)
    auxiliar = int(fun_clr_valor(inicia_memoria_org), 16)
    programa.org_memoria.append(hex(auxiliar).upper()[2::])
    programa.memoria.append(hex(auxiliar).upper()[2::])

def fun_cambio_formato(item_codigo):                                    #
    memoria = int(programa.cambio_formato, 16)          # Memoria en hexa
    if item_codigo == 8:
        temp_aux = 4
    elif item_codigo == 6:
        temp_aux = 3
    elif item_codigo == 4:
        temp_aux = 2
    else:
        temp_aux = 1
    memoria += int(temp_aux)
    programa.cambio_formato = hex(memoria)[2::].upper()

def fun_clr_valor(valor):                           # Quitamos caracteres para el LST
    valor = valor.replace('#','')               # con replace limpiamos la linea
    valor = valor.replace('$','')
    valor = valor.replace(',X','')
    valor = valor.replace(',Y','')
    valor = valor.replace(',','')
    if valor not in programa.var and valor not in programa.etiqueta and valor not in programa.
        salto_etiqueta:
        valor_hex = True
        for letter in valor:
            if letter not in hexdigits:
                valor_hex = False

        if valor_hex:
            if len(valor)==3 or len(valor)==1:                  # Para cadenas de 4bytes
```

```
                while len(valor)!=4:
                    valor = '0'+valor
    return valor

def fun_verifica_long(valor,byts):          # Lanza un error raise al detectar operando incorrecto
    if len(valor) != byts*2:
        raise Errores(1,programa.total_lineas)
    elif not valor.isnumeric() and (valor not in programa.etiqueta or valor not in programa.var):
        raise Errores(6 if valor not in programa.var else 4,programa.total_lineas)

def fun_verifica_etiqueta(tag):                                          #verifica etiquetas
    if tag != "sin_etiqueta" and tag not in programa.etiqueta:
        programa.etiqueta.update({tag:programa.posicion_memoria})    #agrega el valor de la etiqueta
    programa.posicion_memoria += int(len(programa.memoria[-1])/2)

def fun_salto_relativo(X,Y):                                        #hace saltos valuesRELs
    salto_R = X - Y - 2
    if abs(salto_R)>127: # Mas allá de 127 el salto es MUY LEJANO
        raise Errores(2,programa.total_lineas)
    if salto_R<0:
        salto_R = int(bin(salto_R)[3::],2) - (1 << 8)
    salto_R = hex(salto_R)[3 if salto_R < 0 else 2::].upper()
    while len(salto_R)<2:
        salto_R = '0'+salto_R
    return salto_R

def ajuste_De_Linea(lista,first_space):                                 #aquí se ajustan los colores de
     la impresión
    if len(lista)==3 and first_space:
        f_linea = ''.ljust(9)+lista[0].ljust(8)+lista[1].ljust(15)+lista[2]
    elif len(lista) == 3:
        f_linea = lista[0].ljust(9)+lista[1].ljust(8)+lista[2]
    elif len(lista) == 2 and first_space:
        f_linea = ''.ljust(9)+lista[0].ljust(8)+lista[1]
    elif len(lista) == 2:
        f_linea = lista[0].ljust(9)+lista[1]
    elif first_space:
        f_linea = ''.ljust(9)+lista[0]
    else:
        f_linea = lista[0]
    return f_linea
##Esta función regresa la linea sin los comentarios
def quita_comentarios(linea):
    linea = linea.split()                                       #split separa las palabras que
        encuentra en una linea
    for item_codigo in linea:
        if '*' in item_codigo:
            for x in range(len(linea)-1,linea.index(item_codigo)-1,-1): #recorre la linea en un
                intervalo de inicio hasta el comentario
                linea.pop(x)                                     #pop() Devuelve el ultimo valor de
                    la linea
    return linea

    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # ~~~~~~~~~~~ FUNCIONES Y DIRECTIVAS DEL ~~~~~~~~~
    # ~~~~~~~~~~~~~~~~~~ MC68HC11 ~~~~~~~~~~~~~~~~~~~~
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

def BNE(valor,tag,op_name):     # Branch Not Equal - Verifica si NO es igual a 0 y redirecciona si
    no lo es

    if valor == "no_valor":
        valor = tag
    if valor in programa.etiqueta:
        programa.memoria.append(dict_REL[op_name]+fun_salto_relativo(programa.etiqueta[tag],programa
            .posicion_memoria))
```

```
98      elif valor in programa.salto_etiqueta:
99          programa.salto_etiqueta.update({valor:[dict_REL[op_name],programa.total_lineas,programa.
                salto_etiqueta[valor][2]+1,programa.posicion_memoria+2]})
100         programa.memoria.append(valor)
101     else:
102         programa.salto_etiqueta.update({valor:[dict_REL[op_name],programa.total_lineas,1,programa.
                posicion_memoria]})
103         programa.memoria.append(valor)
104     programa.posicion_memoria+=2
105     if tag != "sin_etiqueta" and tag not in programa.etiqueta:
106         programa.etiqueta.update({tag:programa.posicion_memoria})
107
108 def BRCLR(valor,tag,op_name):
109
110     mnem_extra = 0
111     if 'X' in valor or 'Y' in valor:
112         if 'X' in valor:
113             valor = particular[op_name][1]+fun_clr_valor(valor)
114         else:
115             valor = particular[op_name][2]+fun_clr_valor(valor)
116     else:
117         valor = particular[op_name][0]+fun_clr_valor(valor)
118     programa.memoria.append(valor)                              #agregamos el valor de BRCLR a la
            lista
119     programa.posicion_memoria += int(len(programa.memoria[-1])/2)
120     if tag in programa.etiqueta:
121         programa.memoria[-1]=programa.memoria[-1]+fun_salto_relativo(programa.etiqueta[tag],programa
                .posicion_memoria-1)
122         mnem_extra += 1
123     elif tag in programa.salto_etiqueta:
124         programa.salto_etiqueta.update({tag:[valor,programa.total_lineas,programa.salto_etiqueta[
                valor][2]+1,programa.posicion_memoria,op_name]})
125         programa.memoria[-1] = tag
126         mnem_extra += 1
127     elif tag != "sin_etiqueta":
128         programa.salto_etiqueta.update({tag:[valor,programa.total_lineas,1,programa.posicion_memoria
                ,op_name]})
129         programa.memoria[-1] = tag
130     programa.posicion_memoria += mnem_extra
131
132 def EQU(valor,name_archivo,op_name):
133
134     if int(valor.replace('$',''),16)<=int('FF',16):
135         valor = valor[3:]
136     programa.var.update({name_archivo:valor.replace('$','')})             #agregamos el valor a
            nombre en el diccionario
137     programa.memoria.append(valor.replace('$',''))                 #con append se agrega el valor a
            la lista
138
139 def END(valor,tag,op_name):     # Con END marcamos el fin de todas las instrucciones dadas
140
141     programa.final = True
142     programa.codigo_objeto.update({programa.total_lineas:op_name.ljust(8)+(valor if valor != '
            no_valor' else '')})
143
144 def FCB(valor1):
145
146     programa.memoria.append(fun_clr_valor(valor1)+' ')
147
148 def JMP(valor,tag,op_name):           # Para realizar los saltos en memoria
149
150     if valor == "no_valor":
151         valor = tag
152     if valor in programa.salto_etiqueta:
153         programa.salto_etiqueta.update({valor:[dict_EXT[op_name],programa.total_lineas,programa.
                salto_etiqueta[valor][2]+1]})
```

```
154         programa.memoria.append(valor)
155         programa.posicion_memoria += 3
156     elif valor in programa.etiqueta:
157         programa.memoria.append(dict_EXT[op_name]+hex(programa.etiqueta[tag]).upper()[2::])
158         programa.posicion_memoria += 3
159     elif fun_clr_valor(valor).isnumeric() or fun_clr_valor(valor) in programa.var:
160         if (len(fun_clr_valor(valor)) != 2 or len(fun_clr_valor(valor)) != 4) and fun_clr_valor(
                valor) not in programa.var:
161             raise Errores(1,programa.total_lineas)
162         if ',X' in valor:
163             #programa.memoria.append(indiceadox[op_name]+fun_clr_valor(valor))
164             programa.posicion_memoria += 2
165         elif ',Y' in valor:
166             #programa.memoria.append(indiceadoy[op_name]+fun_clr_valor(valor))
167             programa.posicion_memoria += 3
168         elif len(fun_clr_valor(valor)) == 2:
169             programa.memoria.append(dict_DIR[op_name]+fun_clr_valor(valor))
170             programa.posicion_memoria += 2
171         else:
172             valor = programa.var[valor]
173             while len(valor) < 4:
174                 valor = '0'+valor
175             programa.memoria.append(dict_EXT[op_name]+valor)
176             programa.posicion_memoria += 3
177     else:
178         programa.salto_etiqueta.update({valor:[dict_EXT[op_name],programa.total_lineas,1]})
179         programa.memoria.append(valor)
180         programa.posicion_memoria += 3
181 
182 def NOP(valor, tag,op_name):
183 
184     if valor != "no_valor":
185         raise Errores(8,programa.total_lineas)
186     programa.memoria.append(dict_INH[op_name])
187     fun_verifica_etiqueta(tag)
188 
189     # ~~~~~~~~~~ FUNCION DEL MODO DE DIRECCIONAMIENTO PARA ~~~~~~~~
190     # ~~~~~~~~~~~~~~~~~~~~ EL MC68HC1 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
191     # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
192 def LDX(valor,tag,op_name):
193     if fun_clr_valor(valor) in programa.var:
194         if '#' in valor:
195             programa.memoria.append(dict_IMM[op_name]+programa.var[fun_clr_valor(valor)]) #Se manda
196         elif len(programa.var[fun_clr_valor(valor)])==2 and op_name in dict_DIR:
197             programa.memoria.append(dict_DIR[op_name]+programa.var[fun_clr_valor(valor)])
198         elif len(programa.var[fun_clr_valor(valor)])==4:
199             programa.memoria.append(dict_EXT[op_name]+programa.var[fun_clr_valor(valor)])
200         else:
201             programa.memoria.append(dict_EXT[op_name]+programa.var[fun_clr_valor(valor)]) #
                    OPTIMIZACIÓN DE CÓDIGO
202     elif '#' in valor:                                          #dict_IMM
203         valor = fun_clr_valor(valor)
204         if '\'' in valor:
205             programa.memoria.append(dict_IMM[op_name]+hex(ord(valor.replace('\'','')))[2::].upper())
206         elif len(valor) == 2 or len(valor) == 4:
207             programa.memoria.append(dict_IMM[op_name]+valor)
208         else:
209             raise Errores(1,programa.total_lineas)
210     elif ',' in valor:                                          #INDEXADO
211         if 'X' in valor:#X
212             valor = fun_clr_valor(valor)
213             fun_verifica_long(valor, 1)
214             programa.memoria.append(dict_INDX[op_name]+valor)
215         elif 'Y' in valor:#Y
216             valor = fun_clr_valor(valor)
217             fun_verifica_long(valor, 1)
```

```
            programa.memoria.append(dict_INDY[op_name]+valor)
        else:
            raise Errores(7,programa.total_lineas)
    elif '$' in valor:                                          #DIRECTO O EXTENDIDO
        valor = fun_clr_valor(valor)
        if len(valor) == 2:
            programa.memoria.append(dict_DIR[op_name]+valor.replace('$',''))
        elif len(valor) == 4:
            programa.memoria.append(dict_EXT[op_name]+valor.replace('$',''))
        else:
            raise Errores(1,programa.total_lineas)
    elif valor == "no_valor":
        raise Errores(7,programa.total_lineas)
    fun_verifica_etiqueta(tag)
```

Listing 1: Implementacón de las funciones del compilador.

2.3 Lectura del Excel

Esta sección del código (Implementada por los integrantes César y Brian Bautista) Tienen por objetivo la lectura del contenido del archivo Excel en el cual contiene los comandos y mnemonicos del MC68HC11. De este archivo se extraerán los comandos según su OPCODE y para poder manipularlos, se guardarán en un diccionario según el tipo de direccionamiento al que cada comando pertenezca.

```
    # ~~~~~~~~~~~~~~ LEYENDO EL EXCEL ~~~~~~~~~~~~
# Obteniendo los valores desde el Excel INSTRUCCIONES
#Leyendo el contenido del excel
archivo_excel = pd.read_excel('INSTRUCCIONES.xls')
#Guardando la COLUMNA MNEMONICOS
mnemonico_excel = archivo_excel['MNEMONICO'].values

# Obteniendo los valores de los OPCODES
IMM= archivo_excel['OPCODE1'].values
DIR= archivo_excel['OPCODE2'].values
INDX= archivo_excel['OPCODE3'].values
INDY= archivo_excel['OPCODE4'].values
EXT= archivo_excel['OPCODE5'].values
INH= archivo_excel['OPCODE6'].values
REL= archivo_excel['OPCODE7'].values
# Diccionarios para guardar mnemonicos segun su clasificacion
dict_IMM={}
dict_DIR={}
dict_INDX={}
dict_INDY={}
dict_EXT={}
dict_INH={}
dict_REL={}

# Guardando los valores distintos a 'x' en las listas
# Esto se realiza comparando el valor de la columna en
# OPCODE y su correspondiente en la columna MNEMONICO
# del Excel.
# Si el OPCODE es distinto de 'x', entonces al MNEMO
# correspondiente se le asignará el valor de la casilla
# ie: OPCODE1 de IMM, tiene el MNEMO 'adca' con valor 89
i=0
for mnemo in mnemonico_excel:
      if IMM[i]!='x':
            dict_IMM.update({mnemo:str(IMM[i])})
      if DIR[i]!='x':
            dict_DIR.update({mnemo:str(DIR[i])})
      if INDX[i]!='x':
            dict_INDX.update({mnemo:str(INDX[i])})
      if INDY[i]!='x':
```

```
            dict_INDY.update({mnemo:str(INDY[i])})
      if EXT[i]!='x':
            dict_EXT.update({mnemo:str(EXT[i])})
      if INH[i]!='x':
            dict_INH.update({mnemo:str(INH[i])})
      if REL[i]!='x':
            dict_REL.update({mnemo:str(REL[i])})
      i=i+1

# Diccionario particular
particular = {'BCLR':['15','1D','181D'],
              'BRCLR':['13','1F','181F'],
              'BRSET':['12','1E','181E'],
              'BSET':['14','1C','181C'],
              }
```

Listing 2: Lectura del Excel y extracción de datos.

2.4 Implementación de clases

A continuación tenemos la creación de dos clases (Implementadas por el integrante Fernando Maceda), la primera se va a encargar de indicar la cantidad de errores que contiene el programa en caso de que el código se haya escrito de manera incorrecta, mientras que en la segunda clase se definen una serie de variables y métodos que se van a utilizar posteriormente.

```
# Clase que indicara cuantos Errores suceden en el programa
class Errores(BaseException):   # Hereda de BaseException para comportarse como un error.
    def __init__(self,codigo,error_linea,op_name = ''):
        self.error_linea = str(error_linea)                    # Las palabras corruptas se guardan
            en Strings
        self.op_name = op_name
        self.codigo = codigo
        programa.errores +=1

class Program(object):
    # Definiremos variables y métodos a utilizar de la clase
    def __init__(self,name_archivo):               # Método Constructor que tendrá el codigo leido
        self.name_archivo = name_archivo                    # Guarda el nombre del archivo leido, lo
            usa para el HEX y LST
        self.comienzo = False              # Variable que actuara como ORG, marca el inicio del
            programa
        self.final = False                 # Nos marcara como END el fin de lectura del codigo en el
             archivo
        self.inicio_memoria = '0'            # Indicara el inicio del programa en HEXA
        self.cambio_formato = self.inicio_memoria    # Var que nos ayudara a dar formato a los
            archivos
        self.posicion_memoria = 0           # Nos ayudará a dar los saltos en el direccionamiento
            REL
        self.memoria = []                  # Lista que contendrá las direcciones de memoria
            generadas
        self.org_memoria = []              # Lista para las direcciones en HEX
        self.var = {}                      # Diccionario que contendra variables y su nombre
        self.linea_posicion = 0            # Variable para leer linea por linea
        self.etiqueta = {}                 # Diccionario para las etiquetas (tag) y su posicion en
            memoria
        self.codigo_objeto = {}              # Diccionario para construir el codigo objeto
        self.total_lineas = 0              # Variable que cuenta el número total de líneas que tiene
             nuestro codigo
        self.errores = 0                   # Variable que cuenta el número total de erroress que
            sucedieron
        self.salto_etiqueta = {}           # Diccionario que contiene los saltos de etiqueta y a que
             direccion apuntan
```

Listing 3: Implementacón de las clases.

2.5 Excepciones para tratar con los errores del compilador

Como se ha visto en el paradigma de programación orientada a objetos, las excepciones tienen como propósito el ejecutar el programa aún cuando un error se produjo durante la ejecución del programa. En este sentido, estas exepciones (Implementadas por el integrante José Vázquez) van a marcar la cantidad de errores, no de este código hecho en **Python**, sino del código que contiene el archivo ASC. Como en toda compilacón se leerá el archivo línea por línea y de haber errores, el compilador nos lo mostrará. Para esto, antes de la verificación línea por línea y de las excepciones, se creó un diccionario que contiene los mnemonicos del lenguaje del MC68HC11.

```
# Diccionario para los mnemónicos del MC68HC11 Clave:Valor
mnemonico_dict = {
'ABA':NOP,'ABX':NOP,'ABY':NOP, 'ADCA':LDX,'ADCB':LDX,'ADDA':LDX,'ADDB':LDX,'ADDD':LDX,'ANDA':LDX,
'ANDB':LDX,'ASL':LDX,'ASLA':NOP, 'ASLB':NOP,'ASLD':NOP,'ASR':LDX,'ASRA':NOP,'ASRB':NOP,'BCC':BNE,
'BCLR':BRCLR, 'BCS':BNE, 'BEQ':BNE,'BGE':BNE,'BGT':BNE,'BHI':BNE,'BHS':BNE,'BITA':LDX,'BITB':LDX,
'BLE':BNE,'BLO':BNE, 'BLS':BNE, 'BLT':BNE, 'BMI':BNE,'BNE':BNE,'BPL':BNE,'BRA':BNE,'BRCLR':BRCLR,
'BRN':BNE,'BRSET':BRCLR,'BSET':BRCLR,'BSR':BNE,'BVC':BNE,'BVS':BNE,'CBA':NOP,'CLC':NOP,'CLI':NOP,
'CLR':LDX,'CLRA':NOP,'CLRB':NOP,'CLV':NOP, 'CMPA':LDX,'CMPB':LDX,'COM':LDX,'COMA':NOP,'COMB':NOP,
'CPD':LDX,'CPX':LDX,'CPY':LDX,'DAA':NOP, 'DEC':LDX, 'DECA':NOP, 'DECB':NOP, 'DES':NOP, 'DEX':NOP,
'DEY':NOP,'END':END,'EORA':LDX,'EORB':LDX,'EQU':EQU,'FCB':FCB, 'FDIV':NOP, 'IDIV':NOP, 'INC':LDX,
'INCA':NOP,'INCB':NOP,'INS':NOP,'INX':NOP, 'INY':NOP, 'JMP':JMP, 'JSR':JMP,'LDAA':LDX,'LDAB':LDX,
'LDD':LDX, 'LDS':LDX, 'LDX':LDX, 'LDY':LDX, 'LSL':LDX,'LSLA':NOP,'LSLB':NOP,'LSLD':NOP,'LSR':LDX,
'LSRA':NOP, 'LSRB':NOP,'LSRD':NOP,'MUL':NOP,'NEG':LDX,'NEGA':NOP,'NEGB':LDX,'NOP':NOP,'ORAA':LDX,
'ORAB':LDX, 'ORG':INICIO, 'PSHA':NOP, 'PSHB':NOP, 'PSHX':NOP, 'PSHY':NOP, 'PULA':NOP, 'PULB':NOP,
'PULX':NOP, 'PULY':NOP,'ROL':LDX,'ROLA':NOP,'ROLB':NOP,'ROR':LDX,'RORA':NOP,'RORB':NOP,'RTI':NOP,
'RTS':NOP,'SBA':NOP, 'SBCA':LDX, 'SBCB':LDX, 'SEC':NOP,'SEI':NOP,'SEV':NOP,'STAA':LDX,'STAB':LDX,
'STD':LDX,'STOP':NOP,'STS':LDX,'STX':LDX, 'STY':LDX, 'SUBA':LDX, 'SUBB':LDX,'SUBD':LDX,'SWI':NOP,
'TAB':NOP, 'TAP':NOP,'TBA':NOP,'TETS':NOP,'TPA':NOP,'TST':LDX, 'TSTA':NOP, 'TSTB':NOP, 'TSX':NOP,
'TSY':NOP,'TXS':NOP,'TYS':NOP,'WAI':NOP,'XGDX':NOP,'XGDY':NOP,
'aba':NOP,'abx':NOP,'aby':NOP, 'adca':LDX,'adcb':LDX,'adda':LDX,'addb':LDX,'addd':LDX,'anda':LDX,
'andb':LDX,'asl':LDX,'asla':NOP, 'aslb':NOP,'asld':NOP,'asr':LDX,'asra':NOP,'asrb':NOP,'bcc':BNE,
'bclr':BRCLR, 'bcs':BNE, 'beq':BNE,'bge':BNE,'bgt':BNE,'bhi':BNE,'bhs':BNE,'bita':LDX,'bitb':LDX,
'ble':BNE,'blo':BNE, 'bls':BNE, 'blt':BNE, 'bmi':BNE,'bne':BNE,'bpl':BNE,'bra':BNE,'BRCLR':BRCLR,
'brn':BNE,'brset':BRCLR,'bset':BRCLR,'bsr':BNE,'bvc':BNE,'bvs':BNE,'cba':NOP,'clc':NOP,'cli':NOP,
'clr':LDX,'clra':NOP,'clrb':NOP,'clv':NOP, 'cmpa':LDX,'cmpb':LDX,'com':LDX,'coma':NOP,'comb':NOP,
'cpd':LDX,'cpx':LDX,'cpy':LDX,'daa':NOP, 'dec':LDX, 'deca':NOP, 'decb':NOP, 'des':NOP, 'dex':NOP,
'dey':NOP,'end':END,'eora':LDX,'eorb':LDX,'equ':EQU,'fcb':FCB, 'fdiv':NOP, 'idiv':NOP, 'inc':LDX,
'inca':NOP,'incb':NOP,'ins':NOP,'inx':NOP, 'iny':NOP, 'jmp':JMP, 'jsr':JMP,'ldaa':LDX,'ldab':LDX,
'ldd':LDX, 'lds':LDX, 'LDX':LDX, 'ldy':LDX, 'lsl':LDX,'lsla':NOP,'lslb':NOP,'lsld':NOP,'lsr':LDX,
'lsra':NOP, 'lsrb':NOP,'lsrd':NOP,'mul':NOP,'neg':LDX,'nega':NOP,'negb':LDX,'NOP':NOP,'oraa':LDX,
'orab':LDX, 'org':INICIO, 'psha':NOP, 'pshb':NOP, 'pshx':NOP, 'pshy':NOP, 'pula':NOP, 'pulb':NOP,
'pulx':NOP, 'puly':NOP,'rol':LDX,'rola':NOP,'rolb':NOP,'ror':LDX,'rora':NOP,'rorb':NOP,'rti':NOP,
'rts':NOP,'sba':NOP, 'sbca':LDX, 'sbcb':LDX, 'sec':NOP,'sei':NOP,'sev':NOP,'staa':LDX,'stab':LDX,
'std':LDX,'stop':NOP,'sts':LDX,'stx':LDX, 'sty':LDX, 'suba':LDX, 'subb':LDX,'subd':LDX,'swi':NOP,
'tab':NOP, 'tap':NOP,'tba':NOP,'tets':NOP,'tpa':NOP,'tst':LDX, 'tsta':NOP, 'tstb':NOP, 'tsx':NOP,
'tsy':NOP,'txs':NOP,'tys':NOP,'wai':NOP,'xgdx':NOP,'xgdy':NOP
}
```

Listing 4: Diccionario de los mnemonicos.

```
# Leyendo linea por linea
for linea in file_ASC:
    first_space = True if linea[0] == ' ' or linea[0] == '\t' else False  ##Solo va a analizar las
        lineas que tengan un espacio o tabulación al principio
    linea = quita_comentarios(linea)
    programa.total_lineas+=1

    if len(linea)>0:
        programa.linea_posicion+=1                                                    #
            ajuste de linea
        programa.codigo_objeto.update({programa.linea_posicion:ajuste_De_Linea(linea,first_space)})

    try:
        if len(linea)>=2:
            if linea[0] in particular or linea[1] in particular:
                if linea[0] in mnemonico_dict:
```

```
                    mnemonico_dict[linea[0]](linea[1],linea[2] if len(linea) == 3 else "sin_etiqueta
                        ",linea[0])
                else:
                    raise Errores(3,programa.total_lineas,linea[0] if linea[0] in particular else
                        linea[1])
                linea = ''


        if len(linea) == 3:
            if linea[1] not in mnemonico_dict:
                raise Errores(3,programa.total_lineas,linea[1])

            if linea[1] == 'FCB':
                mnemonico_dict[linea[1]](linea[2])
            elif linea[1] == 'fcb':
                mnemonico_dict[linea[1]](linea[2])
            else:
                mnemonico_dict[linea[1]](linea[2], linea[0],linea[1])
        elif len(linea) == 2:
            if linea[0] in mnemonico_dict:
                if linea[1] in programa.etiqueta:
                    mnemonico_dict[linea[0]]("no_valor", linea[1],linea[0])
                elif linea[0] == 'FCB':
                    mnemonico_dict[linea[0]](linea[1])
                elif linea[0] == 'fcb':
                    mnemonico_dict[linea[0]](linea[1])
                else:
                    mnemonico_dict[linea[0]](linea[1], "sin_etiqueta",linea[0])
            elif linea[1] in mnemonico_dict:
                mnemonico_dict[linea[1]]("no_valor", linea[0],linea[1])
            else:
                raise Errores(3,programa.total_lineas,linea[0])
        elif len(linea) == 1:
            if linea[0] in mnemonico_dict and first_space:
                mnemonico_dict[linea[0]]("no_valor", "sin_etiqueta",linea[0])
            elif not first_space:
                programa.etiqueta.update({linea[0]:programa.posicion_memoria})
            else:
                raise Errores(3,programa.total_lineas,linea[0])
    #errores
    except KeyError:
        print("ERROR 007: MAGNITUD DE OPERANDO ERRONEA"+str(programa.total_lineas)+' K')
    except Errores as err:
        if err.codigo == 1:
            print("ERROR 007: MAGNITUD DE OPERANDO ERRONEA "+err.error_linea)
        elif err.codigo == 2:
            print("ERROR 008: SALTO RELATIVO MUY LEJANO"+err.error_linea)
        elif err.codigo == 3:
            print("ERROR 004 MNEMÓNICO"+err.op_name+"\tINEXISTENTE EN LINEA"+err.error_linea)
        elif err.codigo == 4:
            print("ERROR 003 ETIQUETA "+err.op_name+"\tINEXISTENTE EN LINEA"+err.error_linea)
        elif err.codigo == 5:
            print("ERROR 002 VARIABLE "+err.op_name+"INEXISTENTE "+err.error_linea)
        elif err.codigo == 6:
            print("ERROR 001 CONSTANTE"+err.op_name+"INEXISTENTE EN LINEA "+err.error_linea)
        elif err.codigo == 7:
            print("ERROR 005 INSTRUCCIÓN CARECE DE OPERANDO(S)"+err.error_linea)
        elif err.codigo == 8:
            print("ERROR 006 INSTRUCCIÓN NO LLEVA OPERANDO(S)"+err.error_linea)

    #etiquetas salto relativo
for indice,entry in enumerate(programa.memoria):
    if entry in programa.salto_etiqueta and entry in programa.etiqueta:
        if programa.salto_etiqueta[entry][0] in dict_REL.values() or len(programa.salto_etiqueta[
            entry]) == 5:
            try:
```

```
                programa.memoria[indice]= str(programa.salto_etiqueta[entry][0])+fun_salto_relativo(
                    programa.etiqueta[entry],programa.salto_etiqueta[entry][3])
            except Errores:
                print("ERROR 008 SALTO RELATIVO MUY LEJANO"+str(programa.salto_etiqueta[entry][1]))
        else:
            programa.memoria[indice]= str(programa.salto_etiqueta[entry][0])+hex(programa.etiqueta[
                entry])[2::].upper()
        programa.salto_etiqueta[entry][2]+=-1
for key in programa.salto_etiqueta:
    if programa.salto_etiqueta[key][2]>0:
        print("ERROR 003 ETIQUETA"+key+"\tINEXISTENTE EN LINEA"+str(programa.salto_etiqueta[key][1])
            )
        programa.errores += 1
        if programa.memoria.count(key)>1:
            print("\tETIQUETA INEXISTENTE PRESENTE"+str(programa.memoria.count(key))+" veces")
#error 9
if not programa.final:
    programa.errores += 1
    print("ERROR 010: NO SE ENCUENTRA END")
```

Listing 5: Tratado de errores del compilador.

2.6 Generación de los archivos que son producto de la compilación del archivo

Esta sección fue implementada por el integrante Brian Bautista. En esta parte se generan los archivos LST y S19 una vez que el archivo ASC fue compilado. Terminado este proceso se cierran los tres archivos y se le pide al usuario cerrar la ventana, pues si todo salió bien la compilación habrá finalizado.

```
# Cierra los archivos ASC, LST y S19 una vez que terminó de leer / escribir
file_LST.close()
file_s19.close()
file_ASC.close()
#--Genera el archivo .LST
#--Genera el archivo .S19
try:
    print('\n\t Archivo LST ' +programa.name_archivo.replace('.ASC','.lst')+' creado correctamente')
    print('\n\t Archivo S19 ' +programa.name_archivo.replace('.ASC','.s19')+' creado correctamente')
except:
    print('\n\t No se pudo crear: Archivo LST o Archivo S19 . INTENTE COMPILAR DE NUEVO ')
print('\n\t ~~~~~~~~~~~ COMPILACION FINALIZADA, PUEDE CERRAR LA VENTANA ~~~~~~~~~~~')
```

Listing 6: Generación de archivos.

2.7 Ejecución del programa (Petición del archivo ASC)

Esta parte del programa fue implementada por los integrantes Fernando Maceda y José Vázquez. Es lo primero que realizará el programa antes de cualquier cosa, la petición del archivo ASC el cual va a ser el que tiene nuestro código fuente y el que será compilado para después generar los archivos LST y S19.

```
print("\n\t ~~~~~~~~~~~ COMPILADOR BÁSICO PARA EL MC68HC11 ~~~~~~~~~~~")
print("\n")

# Pidiendo el archivo con la extensión .ASC
while True: #Sale del flujo hasta dar el nombre correctamente
    archivo_ASC = input("\n\tIngrese el nombre del archivo *.ASC y presione ENTER: ")
    if archivo_ASC.endswith(".ASC"):
        break
    else:
        print("Archivo inválido, debe tener extensión .ASC; intente de nuevo")
programa = Program(archivo_ASC) # Creamos objeto 'programa' mandando como parámetro 'archivo_ASC'
# A partir de ahora la variable 'programa'

try:
```

```
    file_ASC = open(programa.name_archivo,"r")
    file_LST = open(programa.name_archivo.replace('.ASC','.lst'), "w") # Para el archivo .LST
    file_s19 = open(programa.name_archivo.replace('.ASC','.s19'), "w") # Para el archivo .HEX
except:
    print("Error al tratar de leer el archivo. Vuelva a ejecutar el programa e intente de nuevo.")
    input("Presione ENTER para continuar...")
    raise SystemExit   # Salimos del programa
```

Listing 7: Ejecución del programa.

2.7 Informe de la compilación y ubicación de los errores

En esta parte del programa, lo que se hará es que una vez que el archivo esté compilado, se mandarán a imprimir el número de errores en caso de haberlos y a su vez, la ubicación de la línea en la que se encontró el error. (Implementado por los integrantes César Guevara y Fernando Maceda)

```
print('\n')
print('\n\t ~~~~~~~~~~~~~~~~ INFORMACION DE COMPILACION ~~~~~~~~~~~~~~~~')
print('\n\t Numero de errores: '+str(programa.errores))
if programa.errores > 0:
    input("Presione ENTER para continuar...")
    raise SystemExit

#'''Se compiló el programa en número de línea y el codigo objeto'''
pass_var = False
aumentar_espacios=1


try:
    for indice,item_codigo in enumerate(programa.memoria):
        indice+=aumentar_espacios
        if item_codigo in programa.org_memoria:                # Se obtiene una tupla con item_codigo
        ###Aquí se da el formato al archivo lst
            file_LST.write(str(indice).ljust(4)+':'.ljust(8)+item_codigo.ljust(12)+':'+programa.
                codigo_objeto[indice]+'\n')
            programa.cambio_formato = item_codigo
        else:
            if item_codigo in programa.var.values() and programa.cambio_formato == '0':
                file_LST.write(str(indice).ljust(4)+':'.ljust(8)+item_codigo.zfill(4)+':'.rjust(9)+
                    programa.codigo_objeto[indice]+'\n')
            elif (programa.codigo_objeto[indice][0]!=' ' and programa.codigo_objeto[indice][0]!='\t'
                ) and len(quita_comentarios(programa.codigo_objeto[indice])) == 1:
                while (programa.codigo_objeto[indice][0]!=' ' and programa.codigo_objeto[indice
                    ][0]!='\t') and len(quita_comentarios(programa.codigo_objeto[indice])) == 1:
                    file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+''.
                        ljust(12)+':'+programa.codigo_objeto[indice]+'\n')  #ETIQUETAS
                    indice+=1
                    aumentar_espacios+=1
                if(len(item_codigo)==2):
                    file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
                        item_codigo+')'.ljust(9)+':'+programa.codigo_objeto[indice]+'\n')
                    fun_cambio_formato(len(item_codigo))
                elif(len(item_codigo)==4):
                    file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
                        item_codigo+')'.ljust(7)+':'+programa.codigo_objeto[indice]+'\n')
                    fun_cambio_formato(len(item_codigo))
                elif(len(item_codigo)==8):
                    file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
                        item_codigo+')'.ljust(3)+':'+programa.codigo_objeto[indice]+'\n')
                    fun_cambio_formato(len(item_codigo))

                else:
                    file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
                        item_codigo+')'.ljust(5)+':'+programa.codigo_objeto[indice]+'\n')
```

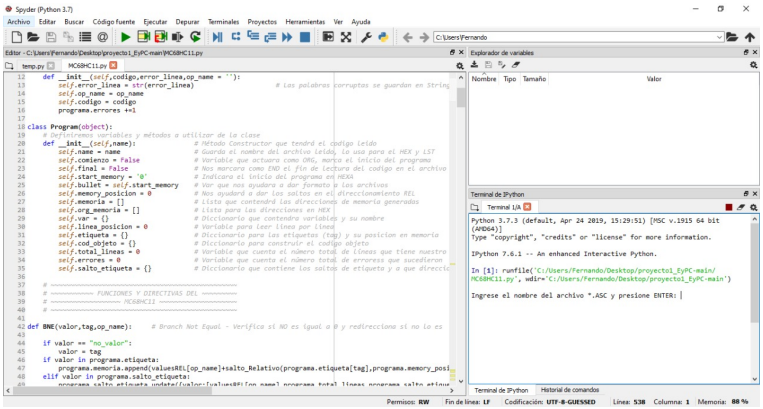
```
                    fun_cambio_formato(len(item_codigo))
            else:
                if(len(item_codigo)==2):
                    file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
                        item_codigo+')'.ljust(9)+':'+programa.codigo_objeto[indice]+'\n')
                    fun_cambio_formato(len(item_codigo))
                elif(len(item_codigo)==4):
                    file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
                        item_codigo+')'.ljust(7)+':'+programa.codigo_objeto[indice]+'\n')
                    fun_cambio_formato(len(item_codigo))
                elif(len(item_codigo)==8):
                    file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
                        item_codigo+')'.ljust(3)+':'+programa.codigo_objeto[indice]+'\n')
                    fun_cambio_formato(len(item_codigo))
                else:
                    file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
                        item_codigo+')'.ljust(5)+':'+programa.codigo_objeto[indice]+'\n')
                    fun_cambio_formato(len(item_codigo))
    indice+=1
    while indice< len(programa.codigo_objeto) and 'END' not in quita_comentarios(programa.
        codigo_objeto[indice-1]):
        if programa.codigo_objeto[indice][0]!=' ' and programa.codigo_objeto[indice][0]!='\t':
          file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+item_codigo.ljust
              (12)+':'+programa.codigo_objeto[indice]+'\n')
        else:
            file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+''.ljust(12)+':
                '+programa.codigo_objeto[indice]+'\n')
        indice+=1
# Indica el error y el numero de linea donde sucedio
except KeyError:
    print('KeyError En linea'+str(indice)+item_codigo+programa.codigo_objeto[indice-1])
# Para la tabla de simbolos presente en el LST
symbol_table = programa.etiqueta.copy()
symbol_table.update(programa.var)
symbol_table.update(programa.salto_etiqueta)
file_LST.write('\nTABLA DE SIMBOLOS, total: '+str(len(symbol_table))+'\n')
for key in sorted(symbol_table):
    if key in programa.etiqueta:
        file_LST.write(key+'\t'+hex(programa.etiqueta[key])[2:].upper()+'\n')
    elif key in programa.salto_etiqueta:
        file_LST.write(key+'\t'+hex(programa.salto_etiqueta[key])[2:].upper()+'\n')
    else:
        file_LST.write(key+'\t'+('' if len(programa.var[key])==4 else '00' )+programa.var[key]+'\n')

cambio_formato = 0
posicion = 1
#Para darle formato al archivo .s19
for item_codigo in programa.memoria:
    if item_codigo in programa.org_memoria:
        posicion = 1
        cambio_formato = item_codigo
        file_s19.write(('\n' if cambio_formato != programa.org_memoria[0] else '<')+cambio_formato+'
            > ')
    elif cambio_formato != 0:
        while posicion<=16 and item_codigo != '':
            file_s19.write(item_codigo[:2]+' ')
            item_codigo = item_codigo.replace(item_codigo[:2],'')
            posicion += 1
        if posicion > 16:
            posicion = 1
            cambio_formato = str(hex(int(cambio_formato,16)+16)[2:].upper()) #upper retorna el
                String en mayus, en este caso valores HEX
            file_s19.write('\n<'+cambio_formato+'> ')
            while item_codigo != '':
                file_s19.write(item_codigo[:2]+' ')
                item_codigo = item_codigo.replace(item_codigo[:2],'')
```

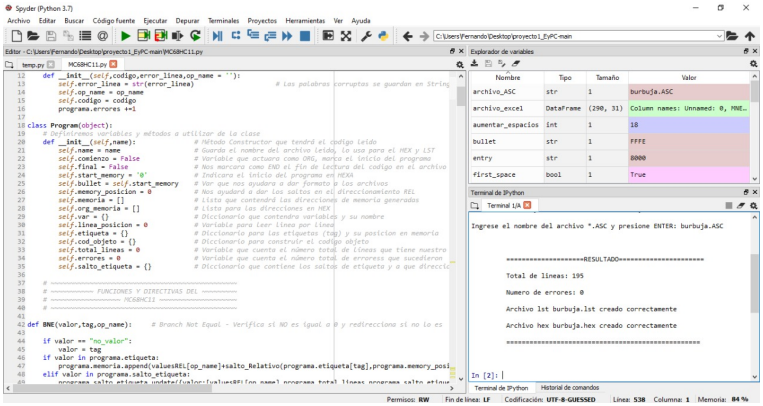
```
97                  posicion += 1
```

Listing 8: Ubicación de los errores.

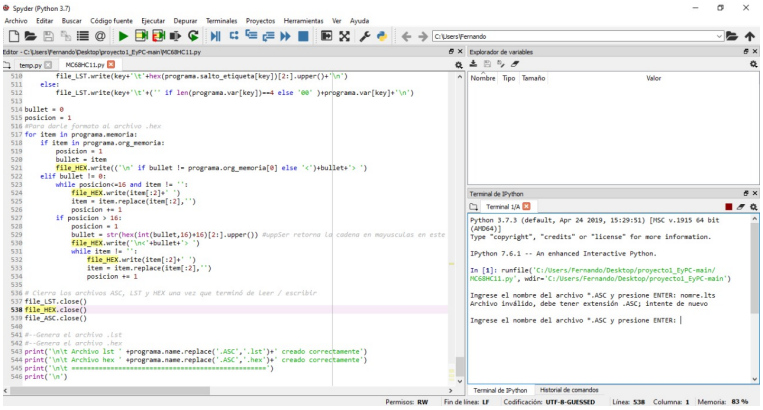
3. Evidencias.



(a) Programa en ejecución



(b) Programa compilado y arrojando los resultados



(c) Error arrojado al no haber escrito correctamente el nombre del archivo ASC

Figura 4: Evidencias de ejecución del compilador del MC68HC11