



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

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:*

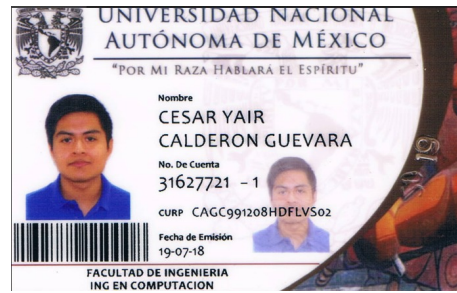
- *Bautista Pérez Brian Jassiel*
- *Calderón Guevara César Yair*
  - *Maceda Patricio Fernando*
  - *Vázquez Flores José Martín*

*Fecha de entrega: 27 de julio de 2021*

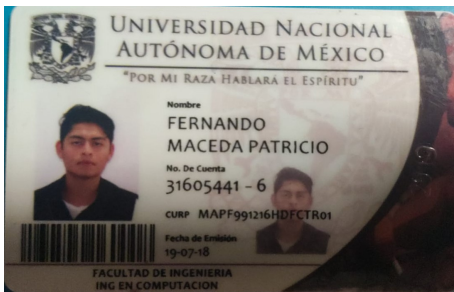
## 1. Identificaciones de los integrantes



(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 implementación 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 código 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 van 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.

```

1 def INICIO(inicia_memoria_org,tag,op_name):          # Llamado ORG, define el inicio del programa
2     if not programa.comienzo:
3         programa.inicio_memoria = fun_clr_valor(inicia_memoria_org)          #inicio programa Memoria en
4             base hexadecimal
5         programa.posicion_memoria = int(programa.inicio_memoria,16) #Nos devuelve al valor del
6             inicio del programa en entero
7         programa.comienzo = True
8         fun_verifica_long(programa.inicio_memoria, 2)
9         auxiliar = int(fun_clr_valor(inicia_memoria_org), 16)
10        programa.org_memoria.append(hex(auxiliar).upper()[2::])
11        programa.memoria.append(hex(auxiliar).upper()[2::])
12
13 def fun_cambio_formato(item_codigo):
14     memoria = int(programa.cambio_formato, 16)          # Memoria en hexa
15     if item_codigo == 8:
16         temp_aux = 4
17     elif item_codigo == 6:
18         temp_aux = 3
19     elif item_codigo == 4:
20         temp_aux = 2
21     else:
22         temp_aux = 1
23     memoria += int(temp_aux)
24     programa.cambio_formato = hex(memoria)[2::].upper()
25
26 def fun_clr_valor(valor):
27     # Quitamos caracteres para el LST
28     # con replace limpiamos la linea
29     valor = valor.replace('#','')
30     valor = valor.replace('$','')
31     valor = valor.replace(',X','')
32     valor = valor.replace(',Y','')
33     valor = valor.replace(',','')
34     if valor not in programa.var and valor not in programa.etiqueta and valor not in programa.
35         salto_etiqueta:
36         valor_hex = True
37         for letter in valor:
38             if letter not in hexdigits:
39                 valor_hex = False
40
41     if valor_hex:
42         if len(valor)==3 or len(valor)==1:          # Para cadenas de 4bytes

```

```

38         while len(valor)!=4:
39             valor = '0'+valor
40     return valor
41
42 def fun_verifica_long(valor,byts):             # Lanza un error raise al detectar operando incorrecto
43     if len(valor) != byts*2:
44         raise Errores(1,programa.total_lineas)
45     elif not valor.isnumeric() and (valor not in programa.etiqueta or valor not in programa.var):
46         raise Errores(6 if valor not in programa.var else 4,programa.total_lineas)
47
48 def fun_verifica_etiqueta(tag):                 #verifica etiquetas
49     if tag != "sin_etiqueta" and tag not in programa.etiqueta:
50         programa.etiqueta.update({tag:programa.posicion_memoria})      #agrega el valor de la etiqueta
51     programa.posicion_memoria += int(len(programa.memoria[-1])/2)
52
53 def fun_salto_relativo(X,Y):                     #hace saltos valuesRELs
54     salto_R = X - Y - 2
55     if abs(salto_R)>127: # Mas allá de 127 el salto es MUY LEJANO
56         raise Errores(2,programa.total_lineas)
57     if salto_R<0:
58         salto_R = int(bin(salto_R)[3:],2) - (1 << 8)
59     salto_R = hex(salto_R)[3 if salto_R < 0 else 2:].upper()
60     while len(salto_R)<2:
61         salto_R = '0'+salto_R
62     return salto_R
63
64 def ajuste_De_Linea(lista,first_space):         #aquí se ajustan los colores de
65     la impresión
66     if len(lista)==3 and first_space:
67         f_linea = ''.ljust(9)+lista[0].ljust(8)+lista[1].ljust(15)+lista[2]
68     elif len(lista) == 3:
69         f_linea = lista[0].ljust(9)+lista[1].ljust(8)+lista[2]
70     elif len(lista) == 2 and first_space:
71         f_linea = ''.ljust(9)+lista[0].ljust(8)+lista[1]
72     elif len(lista) == 2:
73         f_linea = lista[0].ljust(9)+lista[1]
74     elif first_space:
75         f_linea = ''.ljust(9)+lista[0]
76     else:
77         f_linea = lista[0]
78     return f_linea
79 ##Esta función regresa la linea sin los comentarios
80 def quita_comentarios(linea):
81     linea = linea.split()                       #split separa las palabras que
82     encuentra en una linea
83     for item_codigo in linea:
84         if '*' in item_codigo:
85             for x in range(len(linea)-1,linea.index(item_codigo)-1,-1): #recorre la linea en un
86                 intervalo de inicio hasta el comentario
87                 linea.pop(x)                                             #pop() Devuelve el ultimo valor de
88                 la linea
89     return linea
90
91 # ~~~~~
92 # ~~~~~ FUNCIONES Y DIRECTIVAS DEL ~~~~~
93 # ~~~~~ MC68HC11 ~~~~~
94 # ~~~~~
95
96 def BNE(valor,tag,op_name):                     # Branch Not Equal - Verifica si NO es igual a 0 y redirecciona si
97     no lo es
98
99     if valor == "no_valor":
100         valor = tag
101     if valor in programa.etiqueta:
102         programa.memoria.append(dict_REL[op_name]+fun_salto_relativo(programa.etiqueta[tag],programa
103             .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(
161         valor) not in programa.var:
162         raise Errores(1,programa.total_lineas)
163     if ',X' in valor:
164         #programa.memoria.append(indiceadox[op_name]+fun_clr_valor(valor))
165         programa.posicion_memoria += 2
166     elif ',Y' in valor:
167         #programa.memoria.append(indiceadoy[op_name]+fun_clr_valor(valor))
168         programa.posicion_memoria += 3
169     elif len(fun_clr_valor(valor)) == 2:
170         programa.memoria.append(dict_DIR[op_name]+fun_clr_valor(valor))
171         programa.posicion_memoria += 2
172     else:
173         valor = programa.var[valor]
174         while len(valor) < 4:
175             valor = '0'+valor
176         programa.memoria.append(dict_EXT[op_name]+valor)
177         programa.posicion_memoria += 3
178 else:
179     programa.salto_etiqueta.update({valor:[dict_EXT[op_name],programa.total_lineas,1]})
180     programa.memoria.append(valor)
181     programa.posicion_memoria += 3
182
183 def NOP(valor, tag, op_name):
184
185     if valor != "no_valor":
186         raise Errores(8,programa.total_lineas)
187     programa.memoria.append(dict_INH[op_name])
188     fun_verifica_etiqueta(tag)
189
190 # ~~~~~ FUNCION DEL MODO DE DIRECCIONAMIENTO PARA ~~~~~
191 # ~~~~~ EL MC68HC1 ~~~~~
192 # ~~~~~
193 def LDX(valor,tag,op_name):
194     if fun_clr_valor(valor) in programa.var:
195         if '#' in valor:
196             programa.memoria.append(dict_IMM[op_name]+programa.var[fun_clr_valor(valor)]) #Se manda
197         elif len(programa.var[fun_clr_valor(valor)])==2 and op_name in dict_DIR:
198             programa.memoria.append(dict_DIR[op_name]+programa.var[fun_clr_valor(valor)])
199         elif len(programa.var[fun_clr_valor(valor)])==4:
200             programa.memoria.append(dict_EXT[op_name]+programa.var[fun_clr_valor(valor)])
201         else:
202             programa.memoria.append(dict_EXT[op_name]+programa.var[fun_clr_valor(valor)]) #
203             OPTIMIZACIÓN DE CÓDIGO
204     elif '#' in valor:
205         #dict_IMM
206         valor = fun_clr_valor(valor)
207         if '\'' in valor:
208             programa.memoria.append(dict_IMM[op_name]+hex(ord(valor.replace('\'' , '')))[2::].upper())
209         elif len(valor) == 2 or len(valor) == 4:
210             programa.memoria.append(dict_IMM[op_name]+valor)
211         else:
212             raise Errores(1,programa.total_lineas)
213     elif ', ' in valor:
214         #INDEXADO
215         if 'X' in valor:#X
216             valor = fun_clr_valor(valor)
217             fun_verifica_long(valor, 1)
218             programa.memoria.append(dict_INDX[op_name]+valor)
219         elif 'Y' in valor:#Y
220             valor = fun_clr_valor(valor)
221             fun_verifica_long(valor, 1)

```

```

218         programa.memoria.append(dict_INDY[op_name]+valor)
219     else:
220         raise Errores(7,programa.total_lineas)
221 elif '$' in valor:                                     #DIRECTO O EXTENDIDO
222     valor = fun_clr_valor(valor)
223     if len(valor) == 2:
224         programa.memoria.append(dict_DIR[op_name]+valor.replace('$',''))
225     elif len(valor) == 4:
226         programa.memoria.append(dict_EXT[op_name]+valor.replace('$',''))
227     else:
228         raise Errores(1,programa.total_lineas)
229 elif valor == "no_valor":
230     raise Errores(7,programa.total_lineas)
231 fun_verifica_etiqueta(tag)

```

Listing 1: Implementación 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.

```

1     # ~~~~~ LEYENDO EL EXCEL ~~~~~
2     # Obteniendo los valores desde el Excel INSTRUCCIONES
3     #Leyendo el contenido del excel
4     archivo_excel = pd.read_excel('INSTRUCCIONES.xls')
5     #Guardando la COLUMNA MNEMONICOS
6     mnemonico_excel = archivo_excel['MNEMONICO'].values
7
8     # Obteniendo los valores de los OPCODES
9     IMM= archivo_excel['OPCODE1'].values
10    DIR= archivo_excel['OPCODE2'].values
11    INDX= archivo_excel['OPCODE3'].values
12    INDY= archivo_excel['OPCODE4'].values
13    EXT= archivo_excel['OPCODE5'].values
14    INH= archivo_excel['OPCODE6'].values
15    REL= archivo_excel['OPCODE7'].values
16    # Diccionarios para guardar mnemonicos segun su clasificacion
17    dict_IMM={}
18    dict_DIR={}
19    dict_INDX={}
20    dict_INDY={}
21    dict_EXT={}
22    dict_INH={}
23    dict_REL={}
24
25    # Guardando los valores distintos a 'x' en las listas
26    # Esto se realiza comparando el valor de la columna en
27    # OPCODE y su correspondiente en la columna MNEMONICO
28    # del Excel.
29    # Si el OPCODE es distinto de 'x', entonces al MNEMO
30    # correspondiente se le asignará el valor de la casilla
31    # ie: OPCODE1 de IMM, tiene el MNEMO 'adca' con valor 89
32    i=0
33    for mnemo in mnemonico_excel:
34        if IMM[i]!='x':
35            dict_IMM.update({mnemo:str(IMM[i])})
36        if DIR[i]!='x':
37            dict_DIR.update({mnemo:str(DIR[i])})
38        if INDX[i]!='x':
39            dict_INDX.update({mnemo:str(INDX[i])})
40        if INDY[i]!='x':

```

```

41         dict_INDY.update({mnemo:str(INDY[i])})
42     if EXT[i]!='x':
43         dict_EXT.update({mnemo:str(EXT[i])})
44     if INH[i]!='x':
45         dict_INH.update({mnemo:str(INH[i])})
46     if REL[i]!='x':
47         dict_REL.update({mnemo:str(REL[i])})
48     i=i+1
49
50 # Diccionario particular
51 particular = {'BCLR':['15','1D','181D'],
52              'BRCLR':['13','1F','181F'],
53              'BRSET':['12','1E','181E'],
54              'BSET':['14','1C','181C'],
55              }

```

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.

```

1  # Clase que indicara cuantos Errores suceden en el programa
2  class Errores(BaseException): # Hereda de BaseException para comportarse como un error.
3      def __init__(self,codigo,error_linea,op_name = ''):
4          self.error_linea = str(error_linea) # Las palabras corruptas se guardan
5              en Strings
6          self.op_name = op_name
7          self.codigo = codigo
8          programa.erroses +=1
9
10 class Program(object):
11     # Definiremos variables y métodos a utilizar de la clase
12     def __init__(self,name_archivo): # Método Constructor que tendrá el codigo leído
13         self.name_archivo = name_archivo # Guarda el nombre del archivo leído, lo
14             usa para el HEX y LST
15         self.comienzo = False # Variable que actuara como ORG, marca el inicio del
16             programa
17         self.final = False # Nos marcara como END el fin de lectura del codigo en el
18             archivo
19         self.inicio_memoria = '0' # Indicara el inicio del programa en HEXA
20         self.cambio_formato = self.inicio_memoria # Var que nos ayudara a dar formato a los
21             archivos
22         self.posicion_memoria = 0 # Nos ayudará a dar los saltos en el direccionamiento
23             REL
24         self.memoria = [] # Lista que contendrá las direcciones de memoria
25             generadas
26         self.org_memoria = [] # Lista para las direcciones en HEX
27         self.var = {} # Diccionario que contendra variables y su nombre
28         self.linea_posicion = 0 # Variable para leer linea por linea
29         self.etiqueta = {} # Diccionario para las etiquetas (tag) y su posicion en
30             memoria
31         self.codigo_objeto = {} # Diccionario para construir el codigo objeto
32         self.total_lineas = 0 # Variable que cuenta el número total de líneas que tiene
33             nuestro codigo
34         self.erroses = 0 # Variable que cuenta el número total de erroress que
35             sucedieron
36         self.salto_etiqueta = {} # Diccionario que contiene los saltos de etiqueta y a que
37             direccion apuntan

```

Listing 3: Implementación 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 excepciones (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 compilación 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.

```
1 # Diccionario para los mnemónicos del MC68HC11 Clave:Valor
2 mnemonico_dict = {
3     'ABA':NOP, 'ABX':NOP, 'ABY':NOP, 'ADCA':LDX, 'ADCB':LDX, 'ADDA':LDX, 'ADDB':LDX, 'ADD':LDX, 'ANDA':LDX,
4     'ANDB':LDX, 'ASL':LDX, 'ASLA':NOP, 'ASLB':NOP, 'ASLD':NOP, 'ASR':LDX, 'ASRA':NOP, 'ASRB':NOP, 'BCC':BNE,
5     'BCLR':BRCLR, 'BCS':BNE, 'BEQ':BNE, 'BGE':BNE, 'BGT':BNE, 'BHI':BNE, 'BHS':BNE, 'BITA':LDX, 'BITB':LDX,
6     'BLE':BNE, 'BLO':BNE, 'BLS':BNE, 'BLT':BNE, 'BMI':BNE, 'BNE':BNE, 'BPL':BNE, 'BRA':BNE, 'BRCLR':BRCLR,
7     'BRN':BNE, 'BRSET':BRCLR, 'BSET':BRCLR, 'BSR':BNE, 'BVC':BNE, 'BVS':BNE, 'CBA':NOP, 'CLC':NOP, 'CLI':NOP,
8     'CLR':LDX, 'CLRA':NOP, 'CLRB':NOP, 'CLV':NOP, 'CMPA':LDX, 'CMPB':LDX, 'COM':LDX, 'COMA':NOP, 'COMB':NOP,
9     'CPD':LDX, 'CPX':LDX, 'CPY':LDX, 'DAA':NOP, 'DEC':LDX, 'DECA':NOP, 'DECB':NOP, 'DES':NOP, 'DEX':NOP,
10    'DEY':NOP, 'END':END, 'EORA':LDX, 'EORB':LDX, 'EQU':EQU, 'FCB':FCB, 'FDIV':NOP, 'IDIV':NOP, 'INC':LDX,
11    'INCA':NOP, 'INCB':NOP, 'INS':NOP, 'INX':NOP, 'INY':NOP, 'JMP':JMP, 'JSR':JMP, 'LDAA':LDX, 'LDAB':LDX,
12    'LDD':LDX, 'LDS':LDX, 'LDX':LDX, 'LDY':LDX, 'LSL':LDX, 'LSLA':NOP, 'LSLB':NOP, 'LSLD':NOP, 'LSR':LDX,
13    'LSRA':NOP, 'LSRB':NOP, 'LSRD':NOP, 'MUL':NOP, 'NEG':LDX, 'NEGA':NOP, 'NEGB':LDX, 'NOP':NOP, 'ORAA':LDX,
14    'ORAB':LDX, 'ORG':INICIO, 'PSHA':NOP, 'PSHB':NOP, 'PSHX':NOP, 'PSHY':NOP, 'PULA':NOP, 'PULB':NOP,
15    'PULX':NOP, 'PULY':NOP, 'ROL':LDX, 'ROLA':NOP, 'ROLB':NOP, 'ROR':LDX, 'RORA':NOP, 'RORB':NOP, 'RTI':NOP,
16    'RTS':NOP, 'SBA':NOP, 'SBCA':LDX, 'SBCB':LDX, 'SEC':NOP, 'SEI':NOP, 'SEV':NOP, 'STAA':LDX, 'STAB':LDX,
17    'STD':LDX, 'STOP':NOP, 'STS':LDX, 'STX':LDX, 'STY':LDX, 'SUBA':LDX, 'SUBB':LDX, 'SUBD':LDX, 'SWI':NOP,
18    'TAB':NOP, 'TAP':NOP, 'TBA':NOP, 'TETS':NOP, 'TPA':NOP, 'TST':LDX, 'TSTA':NOP, 'TSTB':NOP, 'TSX':NOP,
19    'TSY':NOP, 'TXS':NOP, 'TYS':NOP, 'WAI':NOP, 'XGDX':NOP, 'XG DY':NOP,
20    'aba':NOP, 'abx':NOP, 'aby':NOP, 'adca':LDX, 'adcb':LDX, 'adda':LDX, 'addb':LDX, 'add':LDX, 'anda':LDX,
21    'andb':LDX, 'asl':LDX, 'asla':NOP, 'aslb':NOP, 'asld':NOP, 'asr':LDX, 'asra':NOP, 'asrb':NOP, 'bcc':BNE,
22    'bclr':BRCLR, 'bcs':BNE, 'beq':BNE, 'bge':BNE, 'bgt':BNE, 'bhi':BNE, 'bhs':BNE, 'bita':LDX, 'bitb':LDX,
23    'ble':BNE, 'blo':BNE, 'bls':BNE, 'blt':BNE, 'bmi':BNE, 'bne':BNE, 'bpl':BNE, 'bra':BNE, 'brclr':BRCLR,
24    'brn':BNE, 'brset':BRCLR, 'bset':BRCLR, 'bsr':BNE, 'bvc':BNE, 'bvs':BNE, 'cba':NOP, 'clc':NOP, 'cli':NOP,
25    'clr':LDX, 'clra':NOP, 'clrb':NOP, 'clv':NOP, 'cmpa':LDX, 'cmpb':LDX, 'com':LDX, 'coma':NOP, 'comb':NOP,
26    'cpd':LDX, 'cpx':LDX, 'cpy':LDX, 'daa':NOP, 'dec':LDX, 'deca':NOP, 'decb':NOP, 'des':NOP, 'dex':NOP,
27    'dey':NOP, 'end':END, 'eora':LDX, 'eorb':LDX, 'equ':EQU, 'fcb':FCB, 'fdiv':NOP, 'idiv':NOP, 'inc':LDX,
28    'inca':NOP, 'incb':NOP, 'ins':NOP, 'inx':NOP, 'iny':NOP, 'jmp':JMP, 'jsr':JMP, 'ldaa':LDX, 'ldab':LDX,
29    'ldd':LDX, 'lds':LDX, 'ldx':LDX, 'ldy':LDX, 'lsl':LDX, 'lsla':NOP, 'lslb':NOP, 'lsld':NOP, 'lsr':LDX,
30    'lsra':NOP, 'lsrb':NOP, 'lsrd':NOP, 'mul':NOP, 'neg':LDX, 'nega':NOP, 'negb':LDX, 'nop':NOP, 'oraa':LDX,
31    'orab':LDX, 'org':INICIO, 'psa':NOP, 'psb':NOP, 'psx':NOP, 'psy':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP,
32    'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP,
33    'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP,
34    'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP,
35    'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP,
36    'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP, 'pula':NOP,
37 }
```

Listing 4: Diccionario de los mnemonicos.

```
1 # Leyendo linea por linea
2 for linea in file_ASC:
3     first_space = True if linea[0] == ' ' or linea[0] == '\t' else False ##Solo va a analizar las
4     lineas que tengan un espacio o tabulación al principio
5     linea = quita_comentarios(linea)
6     programa.total_lineas+=1
7
8     if len(linea)>0:
9         programa.linea_posicion+=1
10        ajuste de linea
11        programa.codigo_objeto.update({programa.linea_posicion:ajuste_De_Linea(linea,first_space)})
12
13    try:
14        if len(linea)>=2:
15            if linea[0] in particular or linea[1] in particular:
16                if linea[0] in mnemonico_dict:
```

```

15         mnemonico_dict[linea[0]](linea[1], linea[2] if len(linea) == 3 else "sin_etiqueta", linea[0])
16     else:
17         raise Errores(3, programa.total_lineas, linea[0] if linea[0] in particular else linea[1])
18     linea = ''
19
20
21 if len(linea) == 3:
22     if linea[1] not in mnemonico_dict:
23         raise Errores(3, programa.total_lineas, linea[1])
24
25     if linea[1] == 'FCB':
26         mnemonico_dict[linea[1]](linea[2])
27     elif linea[1] == 'fcb':
28         mnemonico_dict[linea[1]](linea[2])
29     else:
30         mnemonico_dict[linea[1]](linea[2], linea[0], linea[1])
31 elif len(linea) == 2:
32     if linea[0] in mnemonico_dict:
33         if linea[1] in programa.etiqueta:
34             mnemonico_dict[linea[0]]("no_valor", linea[1], linea[0])
35         elif linea[0] == 'FCB':
36             mnemonico_dict[linea[0]](linea[1])
37         elif linea[0] == 'fcb':
38             mnemonico_dict[linea[0]](linea[1])
39         else:
40             mnemonico_dict[linea[0]](linea[1], "sin_etiqueta", linea[0])
41     elif linea[1] in mnemonico_dict:
42         mnemonico_dict[linea[1]]("no_valor", linea[0], linea[1])
43     else:
44         raise Errores(3, programa.total_lineas, linea[0])
45 elif len(linea) == 1:
46     if linea[0] in mnemonico_dict and first_space:
47         mnemonico_dict[linea[0]]("no_valor", "sin_etiqueta", linea[0])
48     elif not first_space:
49         programa.etiqueta.update({linea[0]: programa.posicion_memoria})
50     else:
51         raise Errores(3, programa.total_lineas, linea[0])
52
53 #errores
54 except KeyError:
55     print("ERROR 007: MAGNITUD DE OPERANDO ERRONEA"+str(programa.total_lineas)+' K')
56 except Errores as err:
57     if err.codigo == 1:
58         print("ERROR 007: MAGNITUD DE OPERANDO ERRONEA "+err.error_linea)
59     elif err.codigo == 2:
60         print("ERROR 008: SALTO RELATIVO MUY LEJANO"+err.error_linea)
61     elif err.codigo == 3:
62         print("ERROR 004 MNEMÓNICO"+err.op_name+"\tINEXISTENTE EN LINEA"+err.error_linea)
63     elif err.codigo == 4:
64         print("ERROR 003 ETIQUETA "+err.op_name+"\tINEXISTENTE EN LINEA"+err.error_linea)
65     elif err.codigo == 5:
66         print("ERROR 002 VARIABLE "+err.op_name+"INEXISTENTE "+err.error_linea)
67     elif err.codigo == 6:
68         print("ERROR 001 CONSTANTE"+err.op_name+"INEXISTENTE EN LINEA "+err.error_linea)
69     elif err.codigo == 7:
70         print("ERROR 005 INSTRUCCIÓN CARECE DE OPERANDO(S)+err.error_linea)
71     elif err.codigo == 8:
72         print("ERROR 006 INSTRUCCIÓN NO LLEVA OPERANDO(S)+err.error_linea)
73
74 #etiquetas salto relativo
75 for indice, entry in enumerate(programa.memoria):
76     if entry in programa.salto_etiqueta and entry in programa.etiqueta:
77         if programa.salto_etiqueta[entry][0] in dict_REL.values() or len(programa.salto_etiqueta[entry]) == 5:
78             try:

```

```

78         programa.memoria[indice]= str(programa.salto_etiqueta[entry][0])+fun_salto_relativo(
79             programa.etiqueta[entry],programa.salto_etiqueta[entry][3])
80     except Errores:
81         print("ERROR 008 SALTO RELATIVO MUY LEJANO"+str(programa.salto_etiqueta[entry][1]))
82     else:
83         programa.memoria[indice]= str(programa.salto_etiqueta[entry][0])+hex(programa.etiqueta[
84             entry])[2:].upper()
85         programa.salto_etiqueta[entry][2]+=-1
86 for key in programa.salto_etiqueta:
87     if programa.salto_etiqueta[key][2]>0:
88         print("ERROR 003 ETIQUETA"+key+"\tINEXISTENTE EN LINEA"+str(programa.salto_etiqueta[key][1])
89             )
90         programa.errores += 1
91     if programa.memoria.count(key)>1:
92         print("\tETIQUETA INEXISTENTE PRESENTE"+str(programa.memoria.count(key))+" veces")
93 #error 9
94 if not programa.final:
95     programa.errores += 1
96     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.

```

1  # Cierra los archivos ASC, LST y S19 una vez que terminó de leer / escribir
2  file_LST.close()
3  file_s19.close()
4  file_ASC.close()
5  #--Genera el archivo .LST
6  #--Genera el archivo .S19
7  try:
8      print('\n\t Archivo LST ' +programa.name_archivo.replace('.ASC','.lst')+' creado correctamente')
9      print('\n\t Archivo S19 ' +programa.name_archivo.replace('.ASC','.s19')+' creado correctamente')
10 except:
11     print('\n\t No se pudo crear: Archivo LST o Archivo S19 . INTENTE COMPILAR DE NUEVO ')
12 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.

```

1  print("\n\t ~~~~~~ COMPILADOR BÁSICO PARA EL MC68HC11 ~~~~~~")
2  print("\n")
3
4  # Pidiendo el archivo con la extensión .ASC
5  while True: #Sale del flujo hasta dar el nombre correctamente
6      archivo_ASC = input("\n\tIngrese el nombre del archivo *.ASC y presione ENTER: ")
7      if archivo_ASC.endswith(".ASC"):
8          break
9      else:
10         print("Archivo inválido, debe tener extensión .ASC; intente de nuevo")
11 programa = Program(archivo_ASC) # Creamos objeto 'programa' mandando como parámetro 'archivo_ASC'
12 # A partir de ahora la variable 'programa'
13
14 try:

```

```

15     file_ASC = open(programa.name_archivo, "r")
16     file_LST = open(programa.name_archivo.replace('.ASC', '.lst'), "w") # Para el archivo .LST
17     file_s19 = open(programa.name_archivo.replace('.ASC', '.s19'), "w") # Para el archivo .HEX
18 except:
19     print("Error al tratar de leer el archivo. Vuelva a ejecutar el programa e intente de nuevo.")
20     input("Presione ENTER para continuar...")
21     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)

```

1 print('\n')
2 print('\n\t ~~~~~ INFORMACION DE COMPILACION ~~~~~')
3 print('\n\t Numero de errores: '+str(programa.errores))
4 if programa.errores > 0:
5     input("Presione ENTER para continuar...")
6     raise SystemExit
7
8 '''Se compiló el programa en número de línea y el código objeto'''
9 pass_var = False
10 aumentar_espacios=1
11
12
13 try:
14     for indice,item_codigo in enumerate(programa.memoria):
15         indice+=aumentar_espacios
16         if item_codigo in programa.org_memoria: # Se obtiene una tupla con item_codigo
17             ###Aquí se da el formato al archivo lst
18             file_LST.write(str(indice).ljust(4)+':'.ljust(8)+item_codigo.ljust(12)+':'+programa.
19                 codigo_objeto[indice]+'\\n')
20             programa.cambio_formato = item_codigo
21         else:
22             if item_codigo in programa.var.values() and programa.cambio_formato == '0':
23                 file_LST.write(str(indice).ljust(4)+':'.ljust(8)+item_codigo.zfill(4)+':'.rjust(9)+
24                     programa.codigo_objeto[indice]+'\\n')
25             elif (programa.codigo_objeto[indice][0]!=' ' and programa.codigo_objeto[indice][0]!='\\t'
26                 ) and len(quita_comentarios(programa.codigo_objeto[indice])) == 1:
27                 while (programa.codigo_objeto[indice][0]!=' ' and programa.codigo_objeto[indice]
28                     ][0]!='\\t') and len(quita_comentarios(programa.codigo_objeto[indice])) == 1:
29                     file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
30                         ljust(12)+':'+programa.codigo_objeto[indice]+'\\n') #ETIQUETAS
31                     indice+=1
32                     aumentar_espacios+=1
33             if(len(item_codigo)==2):
34                 file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
35                     item_codigo+')'.ljust(9)+':'+programa.codigo_objeto[indice]+'\\n')
36                 fun_cambio_formato(len(item_codigo))
37             elif(len(item_codigo)==4):
38                 file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
39                     item_codigo+')'.ljust(7)+':'+programa.codigo_objeto[indice]+'\\n')
40                 fun_cambio_formato(len(item_codigo))
41             elif(len(item_codigo)==8):
42                 file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
43                     item_codigo+')'.ljust(3)+':'+programa.codigo_objeto[indice]+'\\n')
44                 fun_cambio_formato(len(item_codigo))
45
46         else:
47             file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
48                 item_codigo+')'.ljust(5)+':'+programa.codigo_objeto[indice]+'\\n')

```

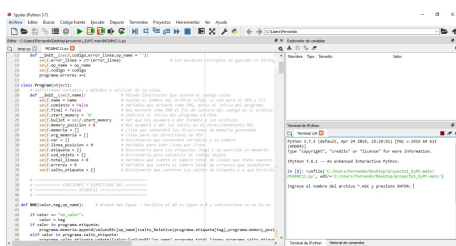
```

40         fun_cambio_formato(len(item_codigo))
41     else:
42         if (len(item_codigo)==2):
43             file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
44                 item_codigo+')'.ljust(9)+':'+programa.codigo_objeto[indice]+'\\n')
45             fun_cambio_formato(len(item_codigo))
46         elif (len(item_codigo)==4):
47             file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
48                 item_codigo+')'.ljust(7)+':'+programa.codigo_objeto[indice]+'\\n')
49             fun_cambio_formato(len(item_codigo))
50         elif (len(item_codigo)==8):
51             file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
52                 item_codigo+')'.ljust(3)+':'+programa.codigo_objeto[indice]+'\\n')
53             fun_cambio_formato(len(item_codigo))
54         else:
55             file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+'('+
56                 item_codigo+')'.ljust(5)+':'+programa.codigo_objeto[indice]+'\\n')
57             fun_cambio_formato(len(item_codigo))
58     indice+=1
59     while indice< len(programa.codigo_objeto) and 'END' not in quita_comentarios(programa.
60         codigo_objeto[indice-1]):
61         if programa.codigo_objeto[indice][0]!=' ' and programa.codigo_objeto[indice][0]!='\t':
62             file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+item_codigo.ljust
63                 (12)+':'+programa.codigo_objeto[indice]+'\\n')
64         else:
65             file_LST.write(str(indice).ljust(4)+':'+programa.cambio_formato.ljust(7)+''.ljust(12)+':
66                 '+programa.codigo_objeto[indice]+'\\n')
67         indice+=1
68 # Indica el error y el numero de linea donde sucedio
69 except KeyError:
70     print('KeyError En linea'+str(indice)+item_codigo+programa.codigo_objeto[indice-1])
71 # Para la tabla de simbolos presente en el LST
72 symbol_table = programa.etiqueta.copy()
73 symbol_table.update(programa.var)
74 symbol_table.update(programa.salto_etiqueta)
75 file_LST.write('\\nTABLA DE SIMBOLOS, total: '+str(len(symbol_table))+ '\\n')
76 for key in sorted(symbol_table):
77     if key in programa.etiqueta:
78         file_LST.write(key+'\\t'+hex(programa.etiqueta[key])[2:].upper()+ '\\n')
79     elif key in programa.salto_etiqueta:
80         file_LST.write(key+'\\t'+hex(programa.salto_etiqueta[key])[2:].upper()+ '\\n')
81     else:
82         file_LST.write(key+'\\t'+(' ' if len(programa.var[key])==4 else '00' )+programa.var[key]+'\\n')
83
84 cambio_formato = 0
85 posicion = 1
86 #Para darle formato al archivo .s19
87 for item_codigo in programa.memoria:
88     if item_codigo in programa.org_memoria:
89         posicion = 1
90         cambio_formato = item_codigo
91         file_s19.write(('\\n' if cambio_formato != programa.org_memoria[0] else '<')+cambio_formato+'
92             > ')
93     elif cambio_formato != 0:
94         while posicion<=16 and item_codigo != '':
95             file_s19.write(item_codigo[:2]+' ')
96             item_codigo = item_codigo.replace(item_codigo[:2], '')
97             posicion += 1
98         if posicion > 16:
99             posicion = 1
100             cambio_formato = str(hex(int(cambio_formato,16)+16)[2:].upper()) #upper retorna el
101                 String en mayus, en este caso valores HEX
102             file_s19.write('\\n<'+cambio_formato+'> ')
103             while item_codigo != '':
104                 file_s19.write(item_codigo[:2]+' ')
105                 item_codigo = item_codigo.replace(item_codigo[:2], '')

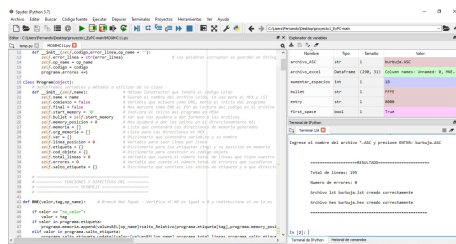
```

Listing 8: Ubicación de los errores.

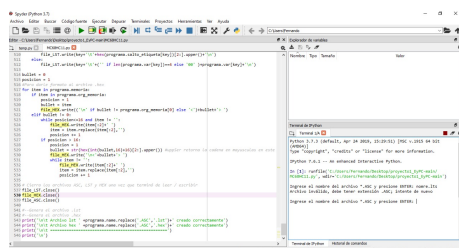
### 3. Evidencias.



(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