
Estructura y Programación de Computadoras

M. en I. Pedro Ignacio Rincón Gómez

Primer Proyecto.



Compilador básico para microprocesador MC68HC11

Meza Peña Augusto.
Moreno Tagle Raphael Iván.
Muñoz Alvarez Rosa María Yolotzin.

April 21, 2015

• Análisis de requerimientos.

Windows, Mac OSX o cualquier otra plataforma con Python 3.4.2 (o anterior aunque se recomienda la versión más reciente) instalado.

<https://www.python.org/downloads/>

Se recomienda Pycharm para la fácil manipulación y ejecución del código.

<https://www.jetbrains.com/pycharm/download/>

• Criterios de Diseño

Desde el inicio la idea central fue que el proyecto se desarrollara en Python (ver. 3.4), ya que es un lenguaje de programación de alto nivel de fácil uso, lo explícito de sus funciones, semántica y sintaxis hacen que se pueda lograr instrucciones complejas y uso de clases entre otras operaciones en pocas líneas de código.

Características.

Código de fácil lectura que permite comprender como se lleva a cabo la ejecución de las diferentes funciones del programa para realizar la compilación.

Completamente portable, lo que significa que no importa en que sistema operativo se ejecute, siguiendo el estándar ANSI siempre funcionará de forma idéntica siempre y cuando se tenga instalado la versión más reciente de [Python](#).

Compilaciones continuas y sin interrupciones con solo indicar el nombre del archivo con extensión .asc generando un archivo .lst y .hex.

```
/Library/Frameworks/Python.framework/Versions/3.4/bin/python3.4
"/Users/Ivan/Documents/4th Semester/Compilador/Main.py"
Archivo a compilar(.asc): EJEMPLO.asc

De 503 líneas de código el compilador detectó 0 errores

Archivo EJEMPLO.lst generado correctamente
Archivo EJEMPLO.hex generado correctamente

Process finished with exit code 0
```

```
EJEMPLO.hex
8000: 8E 00 FF 86 00 B7 10 01 86 FF B7 10 03 86 00 B7
8010: 10 02 CC 30 2C B7 10 2B F7 10 2D 86 00 B7 10 2C
8020: 86 FE B7 10 09 86 04 B7 10 3C 86 00 06 B0 80 CD
8030: 7F 00 11 7F 00 10 7F 00 0F 7F 00 0E 86 3F 97 0C
8040: 96 0C 81 3F 27 FA D6 0E 26 F5 81 73 26 EC 97 0E
8050: 7E 80 3C D6 0F 26 EB 81 54 27 F6 81 53 27 D9 7F
8060: 00 0E 7E 80 3C 97 0F 7E 80 3C D6 10 26 EB 81 41
8070: 27 F6 81 53 26 BA 7F 00 0F 7E 80 3C 97 10 7E 80
8080: 3C D6 11 26 E9 81 52 27 F3 81 53 26 9F 7F 00 10
8090: 7F 00 0F 7E 80 3C 97 11 7E 80 3C 81 54 27 F8 81
80A0: 53 26 86 7F 00 11 7F 00 10 7F 00 0F 7E 80 3C 86
80B0: 45 97 19 86 58 97 1A 86 49 97 1B 86 54 97 1C 86
80C0: 4F 97 1D 7E 80 2D 7F 00 0D 7F 10 02 86 F4 97 18
80D0: CE 10 24 1C 00 4B 0E 10 26 1C 00 50 7F 10 27
80E0: 86 10 97 1B 86 68 97 1C 18 CE 10 68 86 16 97 15
80F0: 86 53 97 16 7F 00 19 7F 00 1A 39 96 19 D6 1A 18
8100: 8F 18 08 18 8F B0 82 48 B0 81 7D B0 81 6D 96 18
8110: D6 1C 18 8F B0 82 48 B0 81 7D B0 81 6D 96 18
8120: B0 81 6D 18 08 4F 18 E6 00 B0 82 58 B0 81 9C B0
8130: 81 6D 18 08 4F 18 E6 00 B0 82 58 B0 81 9C B0 81
8140: 6D 18 08 4F 18 E6 00 B0 82 58 B0 81 9C B0 81 6D
8150: 18 08 4F 18 E6 00 B0 82 58 B0 81 9C 18 8F 97 18
8160: D7 1C B0 81 CA 39 CE 10 2E 86 10 2E 86 2C B7 10
8170: 2F 1F 00 80 FC 39 CE 10 2E 86 10 2E 96 0E 88 30
8180: B7 10 2F 1F 00 80 FC B6 10 2E 96 0F 88 30 B7 10
8190: 2F 1F 00 80 FC CE 10 2E 86 10 2E 96 10 88 30 B7
81A0: 10 2F 1F 00 80 FC B6 10 2E 96 11 88 30 B7 10 2F
81B0: 1F 00 80 FC B6 10 2E 96 12 88 30 B7 10 2F 1F 00
81C0: 80 FC 39 CE 10 2E 86 10 2E 86 0A B7 10 2F 1F 00
81D0: 80 FC B6 10 2E 86 0D B7 10 2F 1F 00 80 FC 39 CE
```

```
EJEMPLO.lst
1: 1026 : PACTL EQU $1026
2: 1027 : PACNT EQU $1027
3: 1030 : ADCTL EQU $1030
4: 1031 : ADR1 EQU $1031
5: 1032 : ADR2 EQU $1032
6: 1033 : ADR3 EQU $1033
7: 1034 : ADR4 EQU $1034
8: 1039 : OPT10N EQU $1039
9: 1000 : PORTA EQU $1000
10: 1008 : PORTD EQU $1008
11: 100A : PORTE EQU $100A
12: 1002 : PORTG EQU $1002
13: 1001 : DDRA EQU $1001
14: 1009 : DDRD EQU $1009
15: 1003 : DDRG EQU $1003
16: 1024 : TMSK2 EQU $1024
17: 1025 : TFLG2 EQU $1025
18: 102F : SCDR EQU $102F
19: 102D : SCCR2 EQU $102D
20: 102E : SCSR EQU $102E
21: 102C : SCCR1 EQU $102C
22: 102B : BAUD EQU $102B
23: 103C : HPRI0 EQU $103C
24: 1028 : SPCR EQU $1028
25: 1050 : CSCTL EQU $1050
26: 1038 : OPT2 EQU $1038
27: 000C : ORDEN EQU $000C
28: 000D : SEG EQU $000D
29: 000E : U1 EQU $000E
30: 000F : U2 EQU $000F
```

- Código del programa.

```
from Database import *
from string import hexdigits
class Program(object):
    def __init__(self, name):
        self.start_memory = '0'
        self.name = name
        self.variables = {}
        self.memory = []
        self.mem_position = 0
        self.line_position = 0
        self.tags = {}
        self.started = False
        self.finished = False
        self.erroses = 0
        self.jump_tags = {}
        self.bullet = self.start_memory
        self.num_bullet = 0
        self.codeline = {}
        self.totallines = 0
        self.org_memory = []
        self.num_code = 0

class Errores(BaseException):
    def __init__(self, code, err_line, op_name = ''):
        self.code = code
        self.err_line = str(err_line)
        self.op_name = op_name
        programa.erroses +=1

def set_bullet(item):
    memory = int(programa.bullet, 16)
    if item == 8:
        aux = 4
    elif item == 6:
        aux = 3
    elif item == 4:
        aux = 2
    else:
        aux = 1
    memory += int(aux)
    programa.bullet = hex(memory)[2:].upper()

def lexer(line):
    line = line.split()
    for item in line:
        if '*' in item:
            for x in range(len(line)-1, line.index(item)-1, -1):
                line.pop(x)
    return line

def clean_value(value):
    value = value.replace('#', '')
    value = value.replace('$', '')
    value = value.replace('X', '')
    value = value.replace('Y', '')
    value = value.replace(',', '')
    if value not in programa.variables and value not in programa.tags and value not in programa.jump_tags:
        is_value_hex = True
        for letter in value:
            if letter not in hexdigits:
                is_value_hex = False

        if is_value_hex:
            if len(value)==3 or len(value)==1: # vi en los programas que siempre "redondea" a 4 bytes
                while len(value)!=4:
                    value = '0'+value
    return value

def relative_jump(a,b):
    jump = a - b - 2
    if abs(jump)>127:
        raise Errores(2, programa.totallines)
    if jump<0:
        jump = int(bin(jump)[3:], 2) - (1 << 8)
    jump = hex(jump)[3] if jump < 0 else 2:~.upper()
    while len(jump)<2:
        jump = '0'+jump
    return jump
```

```

def format_line(list, firstspace):
    if len(list)==3 and first_space:
        fline = ''.ljust(9)+list[0].ljust(8)+list[1].ljust(15)+list[2]
    elif len(list) == 3:
        fline = list[0].ljust(9)+list[1].ljust(8)+list[2]
    elif len(list) == 2 and first_space:
        fline = ''.ljust(9)+list[0].ljust(8)+list[1]
    elif len(list) == 2:
        fline = list[0].ljust(9)+list[1]
    elif first_space:
        fline = ''.ljust(9)+list[0]
    else:
        fline = list[0]
    return fline

def verify_length(value, byts):
    if len(value) != byts*2:
        raise Errores(1, programa.totallines)
    elif not value.isnumeric() and (value not in programa.tags or value not in programa.variables):
        raise Errores(6 if value not in programa.variables else 4, programa.totallines)

def check_tags(tag):
    if tag != "no_tag" and tag not in programa.tags:
        programa.tags.update({tag: programa.mem_position})
    programa.mem_position += int(len(programa.memory[-1])/2)

def INICIO(start_memory_define, tag, op_name): # valor que toma ORG en memoria se usa en bullets.
    if not programa.started:
        programa.start_memory = clean_value(start_memory_define)
        programa.mem_position = int(programa.start_memory, 16)
        programa.started = True
    verify_length(programa.start_memory, 2)
    aux = int(clean_value(start_memory_define), 16)
    programa.org_memory.append(hex(aux).upper()[2::])
    programa.memory.append(hex(aux).upper()[2::])

def LDX(value, tag, op_name):
    #modos de direccionamiento
    if clean_value(value) in programa.variables: #variable extendido
        if '#' in value:
            programa.memory.append(inmediato[op_name]+programa.variables[clean_value(value)])
        elif len(programa.variables[clean_value(value)])==2 and op_name in directo:
            programa.memory.append(directo[op_name]+programa.variables[clean_value(value)])
        elif len(programa.variables[clean_value(value)])==4:
            programa.memory.append(extendido[op_name]+programa.variables[clean_value(value)])
        else:
            programa.memory.append(extendido[op_name]+'00'+programa.variables[clean_value(value)])
        # lo mismo que habias puesto
    elif '#' in value: #inmediato
        value = clean_value(value)
        if '\\' in value:
            programa.memory.append(inmediato[op_name]+hex(ord(value.replace('\\', '')))[2::].upper())
        elif len(value) == 2 or len(value) == 4:
            programa.memory.append(inmediato[op_name]+value)
        else:
            raise Errores(1, programa.totallines)
    elif ',' in value: #indexado
        if 'X' in value: #en X
            value = clean_value(value)
            verify_length(value, 1)
            programa.memory.append(indexadox[op_name]+value)
        elif 'Y' in value: #en Y
            value = clean_value(value)
            verify_length(value, 1)
            programa.memory.append(indexadoy[op_name]+value)
        else:
            raise Errores(7, programa.totallines)
    elif '$' in value: #directo o extendido
        value = clean_value(value) # si el operando es de 12 con el clean_value se hace de 16
        if len(value) == 2:
            programa.memory.append(directo[op_name]+value.replace('$', ''))
        elif len(value) == 4:
            programa.memory.append(extendido[op_name]+value.replace('$', ''))
        else:
            raise Errores(1, programa.totallines)
    elif value == "no_value":
        raise Errores(7, programa.totallines)
    #etiquetas
    check_tags(tag)

```

```

def EQU(value,name,op_name):
    if int(value.replace('$',''),16)<=int('FF',16):
        value = value[3:]
    programa.variables.update({name:value.replace('$','')})
    programa.memory.append(value.replace('$',''))

def NOP(value, tag,op_name):
    if value != "no_value":
        raise Errores(8,programa.totallines)
    programa.memory.append(inherente[op_name])
    check_tags(tag)

def BRCLR(value,tag,op_name):
    mem_add = 0
    if 'X' in value or 'Y' in value:
        if 'X' in value:
            value = especiales[op_name][1]+clean_value(value)
        else:
            value = especiales[op_name][2]+clean_value(value)
    else:
        value = especiales[op_name][0]+clean_value(value)
    programa.memory.append(value)
    programa.mem_position += int(len(programa.memory[-1])/2)
    if tag in programa.tags:
        programa.memory[-1]=programa.memory[-1]+relative_jump(programa.tags[tag],programa.mem_position-1)
        mem_add += 1
    elif tag in programa.jump_tags:
        programa.jump_tags.update({tag:[value,programa.totallines,programa.jump_tags[value]
[2]+1,programa.mem_position,op_name]})
        programa.memory[-1] = tag
        mem_add += 1
    elif tag != "no_tag":
        programa.jump_tags.update({tag:[value,programa.totallines,1,programa.mem_position,op_name]})
        programa.memory[-1] = tag
        programa.mem_position += mem_add

def BNE(value,tag,op_name):
    if value == "no_value":
        value = tag
    if value in programa.tags:
        programa.memory.append(relativo[op_name]+relative_jump(programa.tags[tag],programa.mem_position))
    elif value in programa.jump_tags:
        programa.jump_tags.update({value:[relativo[op_name],programa.totallines,programa.jump_tags[value]
[2]+1,programa.mem_position+2]})
        programa.memory.append(value)
    else:
        programa.jump_tags.update({value:[relativo[op_name],programa.totallines,1,programa.mem_position]})
        programa.memory.append(value)
    programa.mem_position+=2
    if tag != "no_tag" and tag not in programa.tags:
        programa.tags.update({tag:programa.mem_position})

def FCB(value1):
    programa.memory.append(clean_value(value1)+' ')

def JMP(value,tag,op_name):
    if value == "no_value":
        value = tag
    if value in programa.jump_tags:
        programa.jump_tags.update({value:[extendido[op_name],programa.totallines,programa.jump_tags[value][2]+1]})
        programa.memory.append(value)
        programa.mem_position += 3
    elif value in programa.tags:
        programa.memory.append(extendido[op_name]+hex(programa.tags[tag]).upper()[2::])
        programa.mem_position += 3
    elif clean_value(value).isnumeric() or clean_value(value) in programa.variables:
        if (len(clean_value(value)) != 2 or len(clean_value(value)) != 4) and clean_value(value) not in
programa.variables:
            raise Errores(1,programa.totallines)
        if ',X' in value:
            programa.memory.append(indexadox[op_name]+clean_value(value))
            programa.mem_position += 2
        elif ',Y' in value:
            programa.memory.append(indexadoy[op_name]+clean_value(value))
            programa.mem_position += 3
        elif len(clean_value(value)) == 2:
            programa.memory.append(directo[op_name]+clean_value(value))
            programa.mem_position += 2
        else:
            value = programa.variables[value]
            while len(value) < 4:
                value = '0'+value
            programa.memory.append(extendido[op_name]+value)
            programa.mem_position += 3
    else:
        programa.jump_tags.update({value:[extendido[op_name],programa.totallines,1]})
        programa.memory.append(value)
        programa.mem_position += 3

```

```

def END(value,tag,op_name):
    programa.finished = True
    programa.codeline.update({programa.totallines:op_name.ljust(8)+(value if value != 'no_value' else '')})

mnemonico={
'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,'B
EQ':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':B
NE,'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':N
OP,'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':L
DX,'LSLA':NOP,'LSLB':NOP,'LSLD':NOP,'LSR':LDX,'LSRA':NOP,'LSRB':NOP,'LSRD':NOP,'MUL':NOP,'NEG':LDX,'NEGA':NOP,'NEG
B':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,'ST
X':LDX,'STY':LDX,'SUBA':LDX,'SUBB':LDX,'SUBD':LDX,'SWI':NOP,'TAB':NOP,'TAP':NOP,'TBA':NOP,'TETS':NOP,'TPA':NOP,'TS
T':LDX,'TSTA':NOP,'TSTB':NOP,'TSX':NOP,'TSY':NOP,'TXS':NOP,'TYS':NOP,'WAI':NOP,'XGDX':NOP,'XGDY':NOP}

programa = Program(input('Archivo a compilar(.asc): '))
try:
    file = open(programa.name,"r")
    f = open(programa.name.replace('.asc','.lst'), "w+")
    h = open(programa.name.replace('.asc','.hex'), "w+")
except:
    print('Error en el archivo, verifique su existencia')
    exit(-1)
#leer archivo
for linea in file:
    first_space = True if linea[0] == ' ' or linea[0] == '\t' else False
    linea = lexer(linea)
    programa.totallines+=1
    if len(linea)>0:
        programa.line_position+=1
        programa.codeline.update({programa.line_position:format_line(linea,first_space)})
    try:
        if len(linea)>=2:
            if linea[0] in especiales or linea[1] in especiales:
                if linea[0] in mnemonico:
                    mnemonico[linea[0]](linea[1],linea[2] if len(linea) == 3 else "no_tag",linea[0])
                else:
                    raise Errores(3,programa.totallines,linea[0] if linea[0] in especiales else linea[1])
            linea = ''

        if len(linea) == 3:
            if linea[1] not in mnemonico:
                raise Errores(3,programa.totallines,linea[1])
            if linea[1] == 'FCB':
                mnemonico[linea[1]](linea[2])
            else:
                mnemonico[linea[1]](linea[2], linea[0],linea[1])
        elif len(linea) == 2:
            if linea[0] in mnemonico:
                if linea[1] in programa.tags:
                    mnemonico[linea[0]]("no_value", linea[1],linea[0])
                elif linea[0] == 'FCB':
                    mnemonico[linea[0]](linea[1])
                else:
                    mnemonico[linea[0]](linea[1], "no_tag",linea[0])
            elif linea[1] in mnemonico:
                mnemonico[linea[1]]("no_value", linea[0],linea[1])
            else:
                raise Errores(3,programa.totallines,linea[0])
        elif len(linea) == 1:
            if linea[0] in mnemonico and first_space:
                mnemonico[linea[0]]("no_value", "no_tag",linea[0])
            elif not first_space:
                programa.tags.update({linea[0]:programa.mem_position})
            else:
                raise Errores(3,programa.totallines,linea[0])
    except KeyError:
        print("Magnitud de operando erronea en linea "+str(programa.totallines)+' K')
    except Errores as e:
        if e.code == 1:
            print("Magnitud de operando erronea en linea "+e.err_line)
        elif e.code == 2:
            print("Salto relativo muy lejano en linea "+e.err_line)
        elif e.code == 3:
            print("Mnemonico "+e.op_name+"\tinexistente en linea "+e.err_line)
        elif e.code == 4:
            print("Etiqueta "+e.op_name+"\tinexistente en linea "+e.err_line)
        elif e.code == 5:
            print("Variable "+e.op_name+"inexistente en linea "+e.err_line)
        elif e.code == 6:
            print("Constante "+e.op_name+"inexistente en linea "+e.err_line)
        elif e.code == 7:
            print("Instruccion carece operando en linea "+e.err_line)
        elif e.code == 8:
            print("Instruccion no lleva operando en linea "+e.err_line)
#etiquetas de jump malditas...

```

```

for index,entry in enumerate(programa.memory):
    if entry in programa.jump_tags and entry in programa.tags:
        if programa.jump_tags[entry][0] in relativo.values() or len(programa.jump_tags[entry]) == 5:
            try:
                programa.memory[index]= str(programa.jump_tags[entry][0])
+relative_jump(programa.tags[entry],programa.jump_tags[entry][3])
            except Errores as e:
                print("\tSalto relativo muy lejano en linea "+str(programa.jump_tags[entry][1]))
            else:
                programa.memory[index]= str(programa.jump_tags[entry][0])+hex(programa.tags[entry][2:].upper())
                programa.jump_tags[entry][2]+=-1
for key in programa.jump_tags:
    if programa.jump_tags[key][2]>0:
        print("Etiqueta "+key+"\tinexistente en linea "+str(programa.jump_tags[key][1]))
        programa.errores += 1
        if programa.memory.count(key)>1:
            print("\tAviso: Etiqueta inexistente se presenta "+str(programa.memory.count(key))+" veces")
#error 9
if not programa.finished:
    programa.errores += 1
    print("No se encuentra END")
print('\n\t\tDe '+str(programa.totallines)+' lineas de codigo el compilador detecto '+str(programa.errores)+'
errores\n\n')
if programa.errores > 0:
    exit(1)

'''Programa compilado en no. de linea y codigo objeto'''
pass_variables = False
aumentar_espacios=1
try:
    for index,item in enumerate(programa.memory):
        index+=aumentar_espacios
        if item in programa.org_memory:
            f.write(str(index).ljust(4)+':'.ljust(8)+item.ljust(12)+':'+programa.codeline[index]+'\\n')
            programa.bullet = item
        else:
            if item in programa.variables.values() and programa.bullet == '0':
                f.write(str(index).ljust(4)+':'.ljust(8)+item.zfill(4)+':'.rjust(9)+programa.codeline[index]+'\\n')
            elif (programa.codeline[index][0]!=' ' and programa.codeline[index][0]!='\\t') and
len(lexer(programa.codeline[index])) == 1:
                while (programa.codeline[index][0]!=' ' and programa.codeline[index][0]!='\\t') and
len(lexer(programa.codeline[index])) == 1:

f.write(str(index).ljust(4)+':'+programa.bullet.ljust(7)+''.ljust(12)+':'+programa.codeline[index]+'\\n')
            index+=1
            aumentar_espacios+=1

f.write(str(index).ljust(4)+':'+programa.bullet.ljust(7)+item.ljust(12)+':'+programa.codeline[index]+'\\n')
            set_bullet(len(item))
        else:

f.write(str(index).ljust(4)+':'+programa.bullet.ljust(7)+item.ljust(12)+':'+programa.codeline[index]+'\\n')
            set_bullet(len(item))
            index+=1
            while index< len(programa.codeline) and 'END' not in lexer(programa.codeline[index-1]):
                if programa.codeline[index][0]!=' ' and programa.codeline[index][0]!='\\t':
                    f.write(str(index).ljust(4)+':'+programa.bullet.ljust(7)+item.ljust(12)+':'+programa.codeline[index]
+'\\n')
                else:
                    f.write(str(index).ljust(4)+':'+programa.bullet.ljust(7)+''.ljust(12)+':'+programa.codeline[index]
+'\\n')
                index+=1
except KeyError:
    print('Keyerror in line'+str(index)+item+programa.codeline[index-1])

symbol_table = programa.tags.copy()
symbol_table.update(programa.variables)
symbol_table.update(programa.jump_tags)
f.write('\nTabla de Simbolos, total: '+str(len(symbol_table))+'\\n')
for key in sorted(symbol_table):
    if key in programa.tags:
        f.write(key+'\\t'+hex(programa.tags[key][2:].upper())+'\\n')
    elif key in programa.jump_tags:
        f.write(key+'\\t'+hex(programa.jump_tags[key][2:].upper())+'\\n')
    else:
        f.write(key+'\\t'+(' if len(programa.variables[key])==4 else '00')+programa.variables[key]+'\\n')

bullet = 0
position = 1
for item in programa.memory:
    if item in programa.org_memory:
        position = 1
        bullet = item
        h.write('\\n if bullet != programa.org_memory[0] else '')+bullet+': ')
    elif bullet != 0:
        while position<=16 and item != '':
            h.write(item[:2]+' ')
            item = item.replace(item[:2],'')
            position += 1
        if position > 16:
            position = 1

```

```
bullet = str(hex(int(bullet,16)+16)[2:].upper())
h.write('\n'+bullet+': ')
while item != '':
    h.write(item[:2]+' ')
    item = item.replace(item[:2], '')
    position += 1
```

```
f.close()
h.close()
file.close()
print('Archivo '+programa.name.replace('.asc','.lst')+' generado correctamente')
print('Archivo '+programa.name.replace('.asc','.hex')+' generado correctamente')
```


