Anexo V: Instrucciones

```
        ps = NextState[state];              // INST1
        sindex = xlatcase[T[i]];            // INST2
        if (ps[1])                          // INST3 -> P1
            if (MatchList[state])           // INST4 * P1
                tareas a realizar
                una vez encontrado
        state = ps[2+sindex];               // INST5

ps = NextState[state];
INST1 = 2*MOVr32m + ADDrm + MOVmr
{
1   MOVr32m     reg1=state
2   ADDrm       reg1+=NextState
3   MOVr32m     reg2=*(reg1)    //reg2 = NextState[state]
4   MOVmr       ps=reg2         //ps = NextState[state]
}

sindex = xlatcase[T[i]];
INST2 = 3*MOVr32m + 2*ADDrm + MOVmr
{
1   MOVr32m     reg1=i
2   ADDrm       reg1+=T
3   MOVr32m     reg2=*(reg1)    //reg2 = T[i]
4   ADDrm       reg2+=xlatcase
5   MOVr32m     reg3=*(reg2)    //reg3 = xlatcase[T[i]]
6   MOVmr       sindex=reg3     //sindex = xlatcase[T[i]]
}

if (ps[1])
INST3 = MOVri + MOVr32m + ADDrm + CMPrr
{
1   MOVri       reg1=1
2   ADDrm       reg1+=ps
3   MOVr32m     reg2=*(reg1)    //reg2 = ps[1]
4   CMPrr       ¿reg2 != 0?     //if (ps[1])
}

if (MatchList[state])
INST4 = 2*MOVr32m + ADDrm + CMPrr
{
1   MOVr32m     reg1=state
2   ADDrm       reg1+=MatchList
3   MOVr32m     reg2=*(reg1)    //reg2 = MatchList[state]
4   CMPrr       ¿reg2 != 0?     //if (MatchList[state])
}

state = ps[2+sindex];
INST5 = MOVri + MOVr32m + 2*ADDrm + MOVmr
{
1   MOVri       reg1=2
2   ADDrm       reg1+=sindex    //reg1 = 2+sindex
3   ADDrm       reg1+=ps
4   MOVr32m     reg3=*(reg1)    //reg3 = ps[2+index]
5   MOVmr       state=reg3      //state = ps[2+index]
}
```

Código 1: Instrucción equivalente a la rutina Aho-Corasick

```
if (root->next[xlatcase[T[i]]])                                    // INST6
    leaf = root->next[xlatcase[T[i]]];                             // INST7
    if (leaf->next[xlatcase[T[i+1]]])                              // INST8
        qgbf = (REDB_QGBF *) leaf->next[xlatcase[T[i+1]]];         // INST9
        if (qgbf->numnostr)                                        // INST10a
            return 1;
        if (qgbf->numbfstr)                                        // INST10b
            Cbf
        if (qgbf->numshortstr)                                     // INST10c
            Cl

/*

NOTA:
estructura->memvar[i]
    MOVr32m     reg3=i
    MOVr32m     reg4=estructura
    MOVr32m     reg5=X(memvar)      //-> indica la posición de memvar dentro de
estructura
    ADDrr       reg4+=reg5          //reg4 = (estructura->memvar)
    ADDrr       reg4+=reg3          //reg4 = &(estructura->memvar[i])
    MOVr32m     reg6+=*(reg4)       //reg6 = estructura->memvar[i]
*/

if (root->next[xlatcase[T[i]]])
INST6 = 5*MOVr32m + 3*ADDrm + ADDrr + CMPrr
{
1   MOVr32m     reg1=i
2   ADDrm       reg1+=T
3   MOVr32m     reg2=*(reg1)    //reg2 = T[i]
4   ADDrm       reg2+=xlatcase
5   MOVr32m     reg3=*(reg2)    //reg3 = xlatcase[T[i]]
6   MOVr32m     reg4=X(next)
7   ADDrm       reg4+=root      //reg4 = (root->next)
8   ADDrr       reg4+=reg3      //reg4 = root->next[xlatcase[T[i]]]
9   MOVr32m     reg5=*(reg4)    //reg5 = *(root->next[xlatcase[T[i]]])
10  CMPrr       ¿reg5 != 0?     //if (root->next[xlatcase[T[i]]])
}

leaf = root->next[xlatcase[T[i]]];
INST7 = 4*MOVr32m + 3*ADDrm + ADDrr + MOVmr
{
1   MOVr32m     reg1=i
2   ADDrm       reg1+=T
3   MOVr32m     reg2=*(reg1)    //reg2 = T[i]
4   ADDrm       reg2+=xlatcase
5   MOVr32m     reg3=*(reg2)    //reg3 = xlatcase[T[i]]
6   MOVr32m     reg4=X(next)
7   ADDrm       reg4+=root      //reg4 = (root->next)
8   ADDrr       reg4+=reg3      //reg4 = root->next[xlatcase[T[i]]]
9   MOVmr       leaf=reg4       //leaf = root->next[xlatcase[T[i]]]
}
```


Código 2: Instrucción equivalente a la rutina sigMatch (parte 1/2)

```
if (leaf->next[xlatcase[T[i+1]]])
INST8 = MOVri + 4*MOVr32m + 4*ADDrm + ADDrr + CMPrr
{
1   MOVri       reg1=1
2   ADDrm       reg1+=i
3   ADDrm       reg1+=T
4   MOVr32m     reg2=*(reg1)    //reg2 = T[i+1]
5   ADDrm       reg2+=xlatcase
6   MOVr32m     reg3=*(reg2)    //reg3 = xlatcase[T[i+1]]
7   MOVr32m     reg4=X(next)
8   ADDrm       reg4+=leaf      //reg4 = (leaf->next)
9   ADDrr       reg4+=reg3      //reg4 = leaf->next[xlatcase[T[i+1]]]
10  MOVr32m     reg5=*(reg4)    //reg5 = *(leaf->next[xlatcase[T[i+1]]])
11  CMPrr       ¿reg5 != 0?     //if (leaf->next[xlatcase[T[i+1]]])
}

qgbf = (REDB_QGBF *) leaf->next[xlatcase[T[i+1]]];
INST9 = MOVri + 3*MOVr32m + 4*ADDrm + ADDrr + MOVmr
{
1   MOVri       reg1=1
2   ADDrm       reg1+=i
3   ADDrm       reg1+=T
4   MOVr32m     reg2=*(reg1)    //reg2 = T[i+1]
5   ADDrm       reg2+=xlatcase
6   MOVr32m     reg3=*(reg2)    //reg3 = xlatcase[T[i+1]]
7   MOVr32m     reg4=X(next)
8   ADDrm       reg4+=leaf      //reg4 = (leaf->next)
9   ADDrr       reg4+=reg3      //reg4 = leaf->next[xlatcase[T[i+1]]]
14  MOVmr       qgbf=reg4       //qgbf = leaf->next[xlatcase[T[i+1]]]
}

if (qgbf->numnostr)
INST10a = 2*MOVr32m + ADDrm + CMPrr
{
1   MOVr32m     reg1=X(numnostr)
2   ADDrm       reg1+=qgbf      //reg1 = &(qgbf->numnostr)
3   MOVr32m     reg2=*(reg1)    //reg3 = qgbf->numnostr
4   CMPrr       ¿reg2 != 0?     //if (qgbf->numnostr)
}

if (qgbf->numbfstr)
INST10b = 2*MOVr32m + ADDrm + CMPrr
{
1   MOVr32m     reg1=X(numbfstr)
2   ADDrm       reg1+=qgbf      //reg1 = &(qgbf->numbfstr)
3   MOVr32m     reg2=*(reg1)    //reg3 = qgbf->numbfstr
4   CMPrr       ¿reg2 != 0?     //if (qgbf->numbfstr)
}

if (qgbf->numshortstr)
INST10c = 2*MOVr32m + ADDrm + CMPrr
{
1   MOVr32m     reg1=X(numshortstr)
2   ADDrm       reg1+=qgbf      //reg1 = &(qgbf->numshortstr)
3   MOVr32m     reg2=*(reg1)    //reg3 = qgbf->numshortstr
4   CMPrr       ¿reg2 != 0?     //if (qgbf->numshortstr)
}
```


Código 3: Instrucción equivalente a la rutina sigMatch (parte 2/2)

```
for (j=0, p=i+RB_B; j<RB_BETA; j++, p++)                // INST11+(RB_BETA)*INST12
    bfpayload[j] = xlatcase[payload[p]];                // (RB_BETA)*INST13
bfpayload[RB_BETA] = '\0';                              // INST14
hashXOR = 0;                                            // INST15
for (bi = 0; bi<RB_BETA; bi++)                          // INST16+(RB_BETA)*INST17
    hashXOR ^= *((unsigned int *) (bfpayload+bi));      // (RB_BETA)*INST18
if (qgbf->bfmatrix &&                                   // INST19
   (qgbf->bfmatrix[(hashXOR%                            //
   qgbf->bfsize)/SIZE_BYTE]                             //
   & (1 << ((hashXOR % qgbf->bfsize) % SIZE_BYTE))))    //
    return 1;


for: j=0, p=i+RB_B; {j<RB_BETA (1 iteración)}
INST11 = MOVmi + 2*MOVri + ADDrm + MOVmr + CMPrm
{
1   MOVmi       j=0             //j = 0
2   MOVri       reg1=RB_B
3   ADDrm       reg1+=i         //reg1 = i+RB_B
4   MOVmr       p=reg1          //p = i+RB_B
5   MOVri       reg2=RB_BETA    //reg2 = RB_BETA
6   CMPrm       ¿reg2>j?        //j<RB_BETA (1 iteración)
}

for: {j<RB_BETA; j++, p++ (RB_BETA iteraciones)}
INST12 = 2*INCm + MOVri + CMPrm
{
1   INCm        j++             //j++
2   INCm        p++             //p++
3   MOVri       reg1=RB_BETA    //reg1 = RB_BETA
4   CMPrm       ¿reg1>j?        //j<RB_BETA
}

bfpayload[j] = xlatcase[payload[p]];
INST13 = 4*MOVr32m + 3*ADDrm + MOVmr
{
1   MOVr32m     reg1=p
2   ADDrm       reg1+=payload
3   MOVr32m     reg2=*(reg1)    //reg2 = payload[p]
4   ADDrm       reg2+=xlatcase
5   MOVr32m     reg3=*(reg2)    //reg3 = xlatcase[payload[p]]
6   MOVr32m     reg4=j
7   ADDrm       reg4+=bfpayload
8   MOVmr       *(reg4)=reg3    //bfpayload[j] = xlatcase[payload[p]]
}

bfpayload[RB_BETA] = '\0';
INST14 = MOVri + ADDrm + MOVmi
{
1   MOVri       reg1=RB_BETA
2   ADDrm       reg1+=bfpayload
3   MOVmi       *(reg1)='\0'    //bfpayload[RB_BETA] = '\0'
}

hashXOR = 0;
INST15 = MOVmi
{
1   MOVmi       hashXOR=0       //hashXOR = 0
}

for: bi=0; {bi<RB_BETA (1 iteración)}
INST16 = MOVmi + MOVri + CMPrm
{
1   MOVmi       bi=0            //bi = 0
2   MOVri       reg1=RB_BETA    //reg1 = RB_BETA
3   CMPrm       ¿reg1>bi?       //bi<RB_BETA (1 iteración)
}
```


Código 4: Instrucción equivalente a la rutina BF (parte 1/2)

```
for: {bi<RB_BETA; bi++ (RB_BETA iteraciones)}
INST17 = 2*INCm + MOVri + CMPrm
{
1   INCm        bi++            //bi++
2   MOVri       reg1=RB_BETA    //reg1 = RB_BETA
3   CMPrm       ¿reg1>bi?       //bi<RB_BETA
}

hashXOR ^= *((unsigned int *) (bfpayload+bi));
INST18 = 2*MOVr32m + ADDrr + XORrm + MOVmr
{
1   MOVr32m     reg1=bi
2   ADDrm       reg1+=bfpayload //reg1 = bfpayload+bi
3   MOVr32m     reg2=*(reg1)    //reg2 = *(bfpayload+bi)
4   XORrm       reg2^=hashXOR
5   MOVmr       hashXOR=reg2    //hashXOR ^= *(bfpayload+bi)
}

if (qgbf->bfmatrix &&                                   // INST9
   (qgbf->bfmatrix[(hashXOR%                            //
   qgbf->bfsize)/SIZE_BYTE]                             //
   & (1 << ((hashXOR % qgbf->bfsize) % SIZE_BYTE))))    //
INST19 = 4*MOVr32m + MOVri + 2*ADDrm + 3*CMPrr +
         2*DIVr32/m32 + 2*MOVrr + MOVmi + DECr + 2*ANDrr
{
1   MOVr32m     reg1=X(bfmatrix)
2   ADDrm       reg1+=qgbf          //reg1 = &(qgbf->bfmatrix)
3   MOVr32m     reg2=*(reg1)        //reg2 = qgbf->bfmatrix
4   CMPrr       reg3=¿reg2 != 0?    //if (qgbf->bfmatrix)
5   MOVr32m     reg4=X(bfsize)
6   ADDrm       reg4+=qgbf          //reg4 = qgbf->bfsize
7   MOVr32m     reg5=hashXOR        //reg6 = hashXOR
8   DIVr32/m32  reg5/(*reg4)
9   MOVrr       reg5=(resto)        //reg5 = resto{hashXOR/qgbf->bfsize}
10  MOVrr       reg6=reg5           //reg6 = hashXOR%qgbf->bfsize
11  MOVmi       mem1=SIZE_BYTE
12  DIVr32/m32  reg5/mem1           //reg5 = (hashXOR%qgbf->bfsize)/SIZE_BYTE
13  MOVri       reg7=SIZE_BYTE
14  DECr        reg7
15  ANDrr       reg7&=reg6          //reg7 = (hashXOR%qgbf->bfsize)%SIZE_BYTE
16  ANDrr       reg7&=reg5
17  CMPrr       reg8=¿reg7 != 0?
18  CMPrr       ¿reg3 == reg8?      //if (qgbf->bfmatrix&&((hashXOR%qgbf->bfsize)/
                                    //SIZE_BYTE & (hashXOR%qgbf->bfsize)%SIZE_BYTE))
}
```

[2/2]

Código 5: Instrucción equivalente a la rutina BF (parte 2/2)

```
for (k=0; k<qgbf->numshortstr; k++)                // INST20+(qgbf->numshortstr)*INST21
    flag_detect = 1;                               // (qgbf->numshortstr)*INST22
    for (j=0, p=i+RB_B; j<qgbf->lenshortstr[k]; j++, p++)
                          // (qgbf->numshortstr)*(INST23+(qgbf->lenshortstr[k])*INST24)
        if (xlatcase[payload[p]] != qgbf->shortstr[k][j])
                          // (qgbf->numshortstr)*(qgbf->lenshortstr[k])*INST25
            flag_detect = 0;                       //(qgbf->numshortstr)*INST26
            break;
    if (flag_detect)                               // (qgbf->numshortstr)*INST27
        return 1;

for: k=0; {k<qgbf->numshortstr (1 iteración)}
INST20 = 2*MOVr32m + MOVri + ADDrm + CMPrm
{
1   MOVmi       k=0                 //k = 0
2   MOVr32m     reg1=X(numshortstr)
3   ADDrm       reg1+=qgbf          //reg1 = &(qgbf->numshortstr)
4   MOVr32m     reg2=*(reg1)        //reg2 = qgbf->numshortstr
5   CMPrm       ¿reg2>k?            //k<qgbf->numshortstr (1 iteración)
}

for: {k<qgbf->numshortstr; k++ (qgbf->numshortstr iteraciones)}
INST21 = INCm + 2*MOVr32m + ADDrm + CMPrm
{
1   INCm        k++                 //k++
2   MOVr32m     reg1=X(numshortstr)
3   ADDrm       reg1+=qgbf          //reg1 = &(qgbf->numshortstr)
4   MOVr32m     reg2=*(reg1)        //reg2 = qgbf->numshortstr
5   CMPrm       ¿reg2>k?            //k<qgbf->numshortstr
}

flag_detect = 1;
INST22 = MOVmi
{
1   MOVmi       flag_detect=1   //flag_detect = 1
}

for: j=0, p=i+RB_B; {j<qgbf->lenshortstr[k] (1 iteración)}
INST23 = MOVmi + MOVri + 3*ADDrm + MOVmr + 2*MOVr32m + CMPrm
{
1   MOVmi       j=0                 //j = 0
2   MOVri       reg1=RB_B
3   ADDrm       reg1+=i             //reg1 = i+RB_B
4   MOVmr       p=reg1              //p = i+RB_B
5   MOVr32m     reg2=X(lenshortstr)
6   ADDrm       reg2+=qgbf          //reg2 = (qgbf->lenshortstr)
7   ADDrm       reg2+=k             //reg2 = &(qgbf->lenshortstr[k])
8   MOVr32m     reg3=*(reg2)        //reg3 = qgbf->lenshortstr[k]
9   CMPrm       ¿reg3>j?            //j<qgbf->lenshortstr[k] (1 iteración)
}

for: {j<RB_BETA; j++, p++ (qgbf->lenshortstr[k] iteraciones)}
INST24 = 2*INCm + 2*MOVr32m + 2*ADDrm + CMPrm
{
1   INCm        j++                 //j++
2   INCm        p++                 //p++
3   MOVr32m     reg1=X(lenshortstr)
4   ADDrm       reg1+=qgbf          //reg1=(qgbf->lenshortstr)=&(qgbf->lenshortstr[0])
5   ADDrm       reg1+=k             //reg1 = &(qgbf->lenshortstr[k])
6   MOVr32m     reg2=*(reg1)        //reg2 = qgbf->lenshortstr[k]
7   CMPrm       ¿reg2>j?            //j<qgbf->lenshortstr[k]
}
```

[1/2]

Código 6: Instrucción equivalente a la rutina L (parte 1/2)

```
if (xlatcase[payload[p]] != qgbf->shortstr[k][j])
INST25 = 6*MOVr32m + 5*ADDrm + CMPrr
{
1   MOVr32m     reg1=p
2   ADDrm       reg1+=payload
3   MOVr32m     reg2=*(reg1)        //reg2 = payload[p]
4   ADDrm       reg2+=xlatcase      //reg2 = &(xlatcase[payload[p]])
5   MOVr32m     reg3=*(reg2)        //reg3 = xlatcase[payload[p]]
6   MOVr32m     reg4=X(shortstr)
7   ADDrm       reg4+=qgbf
8   ADDrm       reg4+=k             //reg4 = &(qgbf->shortstr[k])
9   MOVr32m     reg5=*(reg4)        //reg5 = &(qgbf->shortstr[k][0]) = qgbf->shortstr[k]
10  ADDrm       reg5+=j             //reg5 = &(qgbf->shortstr[k][j])
11  MOVr32m     reg6=*(reg5)        //reg6 = qgbf->shortstr[k][j]
12  CMPrr       ¿reg3 != reg6?      //if (xlatcase[payload[p]] != qgbf->shortstr[k][j])
}

flag_detect = 0;
INST26 = MOVmi
{
1   MOVmi       flag_detect=0   //flag_detect = 0
}

if (flag_detect)
INST27 = CMPrm
{
1    CMPrm       ¿0 != flag_detect?  //if (flag_detect)
}
```


Código 7: Instrucción equivalente a la rutina L (parte 2/2)