UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Enginyeria de Sistemes TIC

Màquina de xifrar

Dispositius Programables — Enginyeria de Sistemes TIC

Jordi Bonet Francisco del Águila

22 de novembre de 2022

Índex

1 Objectiu

L'objectiu d'aquesta pràctica és realitzar una màquina per xifrar missatges de text. L'AVR rebrà pel port sèrie un missatge i retornarà pel port serie el missatge xifrat. El sistema de xifrat està basat en una traducció caràcter a caràcter, es a dir, byte a byte. L'algoritme de xifrat es basa en una taula d'equivalència on a cada caràcter de text en clar li correspon un caràcter de text xifrat.

2 Descripció de la màquina de xifrar

Aquest xifrador funciona a nivell de byte, es a dir, transforma un byte en un altre. No hi ha un algoritme de transformació definit, com podria ser sumar (en mòdul 256) un desplaçament al byte d'entrada per donar el byte de sortida. L'algoritme de transformació es basa en una taula on cada possible byte d'entrada està aparellat amb un altre byte de sortida.

Els possibles bytes de *text en clar* són el codi ASCII dels caràcters 'A' a 'Z', 'a' a 'z' i l'espai. L'entrada de dades tant en majúscules com en minúscules s'ha de tractar de la mateixa manera i, per tant, aquest detall s'ha de tenir en compte per definir el programa.

Els possibles bytes de *text xifrat* són caràcters imprimibles de la taula ASCII, de manera que podem usar aplicacions com *picocom* o *cutecom*.

Per simplificar aquest xifrador, el caràcter corresponent a l'espai es deixarà sense transformar. Això indica la mida de les paraules xifrades, cosa que afegeix feblesa al sistema de xifrat. El sistema de xifrat que usa una taula s'anomena mono-alfabètic. Aquests sistemes poden ser *trencats* per un atac estadístic si es disposa d'un text xifrat prou llarg. Tot i això, aquest sistema permetrà fer ús de recursos de l'AVR que no s'han vist fins ara.

3 Nous recursos

3.1 Recursos de l'AVR

En aquesta pràctica es farà ús de la memòria de dades pròpiament dita. En aquesta memòria podem definir les variables que ens puguin interessar, juntament amb la taula de xifrat. També s'utilitzarà l'adreçament indirecte a aquesta memòria de dades.

Per mantenir la compatibilitat amb altres assembladors d'altres màquines, i amb la intenció de que l'assemblador sigui genèric per moltes màquines, l'assemblador del GNU pot definir diferents seccions en el codi. Les més importants són:

text: És on es troba el programa pròpiament dit, per tant el seu contingut són els *opcodes* que formen el programa.

data: És on es troben dades que contenen valors concrets inicialitzats. Aquí és on trobarem, per exemple, la taula de xifrat.

bss: És on es troben dades inicialitzades amb valor 0. Aquí es trobaran les variables generals inicialitzades a 0.

L'aplicació particular d'aquestes seccions a l'arquitectura Harvard de l'AVR té diferents implicacions:

1. La secció *text* és directament el codi. De fet, tots els programes de les pràctiques fetes fins ara estan associats per defecte a la secció text.

2. La secció *data* conté variables que contenen valors concrets. D'una banda aquests valors s'han de poder conservar quan s'apaga i es torna a engegar el dispositiu, fet que obliga a emmagatzemar aquests valors en una memòria no volàtil. D'altra banda, aquestes variables són dades, fet que obliga a emmagatzemar aquestes dades a la memòria de dades (SRAM) que és volàtil. Prescindint de la memòria especial EEPROM, que no tots els dispositius AVR contenen, la manera de compatibilitzar aquest fet és que les variables estiguin en una zona de la memòria de programa de l'AVR (que no és volàtil) i en el moment de l'arrencada del sistema es faci una còpia de la memòria de programa a la de dades (que sí és volàtil). Aquest procés queda automatitzat en l'entorn del GNU-gcc.

3. La secció *bss* conté variables inicialitzades a 0. Com que el valor és conegut, no cal emmagatzemar cap valor en una memòria no volàtil. Però sí que cal que en el moment de l'arrencada el sistema reservi l'espai d'aquestes variables a la memòria de dades i, a més, les fixi a valor 0. Aquest procediment també es fa de manera automàtica.

Per tal que aquestes tasques extres, que afecten a les seccions *data* i *bss*, es realitzin en el moment d'arrencar el sistema, cal que en el codi declarem uns símbols reservats com a globals:

.global ___do_copy_data Això permet que les dades que es troben a la secció *data* siguin copiades de la memòria de programa a la memòria de dades.

.global ___do_clear_bss Això permet que es reservi a la memòria de dades l'espai per a les variables que estan inicialitzades a 0.

3.2 Recursos de l'assemblador

A continuació es descriuen algunes directives d'assemblador útils quan els programes comencen a ser complexos. La descripció detallada d'aquestes directives es troben a [AS].

.section Serveix per definir a quina secció va el codi que ve a continuació. Requereix com a paràmetre el nom de la secció.

.text .data .bss Defineixen el tipus de secció on anirà el codi que ve a continuació.

.include Serveix per incloure just en aquell punt el contingut d'un fitxer extern que es farà servir en el programa.

.skip .space Aquestes directives són equivalents i serveixen per omplir un número de bytes definit pel primer paràmetre amb el contingut definit pel segon paràmetre. Els paràmetres poden ser expressions.

.fill És una directiva, similar a .skip o .space, que també serveix per omplir dades amb valors concrets

.byte Accepta expressions per omplir el resultat en forma de byte. La separació amb coma omple els bytes de forma consecutiva.

.ascii .asciiz Omple de forma consecutiva la memòria amb els caràcters de l'*string* que es dóna com a paràmetre. L'opció *asciiz* acaba amb el byte 0x00 a l'última posició .

.if .else .elseif .endif Serveixen per incloure un tros de codi o un altre en funció d'una condició. La forma més simple de definir una condició és avaluar una expressió basada en la definició d'un símbol i el seu valor. En el programa d'exemple apareixen aquestes directives.

3.2.1 Modificadors

Les adreces de memòria de dades en el ATmega328p són de 16 bits. Per tractar aquestes adreces amb els registres existents de 8 bits tenim la necessitat de diferenciar el byte de més pes i el byte de menys pes. L'assemblador del GNU disposa d'uns modificadors específics per tractar amb els AVR que permeten treballar fàcilment amb les adreces de 16 bits. En el programa d'exemple es veu com s'utilitzen aquests modificadors.

hi8() Permet extreure d'una etiqueta de posició de memòria (16 bits) el byte de major pes.

lo8() Permet extreure d'una etiqueta de posició de memòria (16 bits) el byte de menor pes.

4 Programa d'exemple

El següent programa d'exemple, que només té utilitat didàctica, fa ús de l'adreçament indirecte per accedir a una taula de dues columnes emmagatzemada en memòria. Cada cop que es rep un byte pel port sèrie, es transmet pel port sèrie el contingut de la primera columna fins a la posició SALTA-1 (de la posició 0 a la posició SALTA-1) i, a continuació, el contingut de la segona columna des de la posició SALTA fins al final. Noteu que si el valor de DEBUGAR és

1, allò que realment es transmet no és el contingut de la taula sinó el contingut de la taula +1. Observeu també que els valors de SALTA i DEBUGAR són constants, queden fixats en temps de compilació, i no es poden modificar durant l'execució del codi.

```
.set DDRB_o , 0x4
.equ PORTB_o , 0x5
PORTD_o = 0x0b
DDRD_o = 0x0a

UDR0 = 0xc6
UBRR0H = 0xc5
UBRR0L = 0xc4
UCSR0C = 0xc2
UCSR0B = 0xC1
UCSR0A = 0xC0

DEBUGAR = 0
SALTA = 1


.global main
.global __vector_18

.section .text

/* rutina de transmissió de byte,
el valor a transmetre està al registre r16 */
tx:
    lds  r17,UCSR0A
    sbrs r17,5
    rjmp tx
    sts UDR0,r16
    ret

/* defineixo la rutina d'interrupció
per recepció de byte a la USART */
__vector_18:
    lds r16,UDR0

    ldi r26,lo8(taulac1)
    ldi r27,hi8(taulac1)
sigui: ld r16,X+
.if DEBUGAR == 1
    inc r16
.endif
    call tx
    cpi r26,lo8(taulac1+SALTA)
    brne sigui
```

```
    ldi r26,lo8(taulac2+SALTA)
    ldi r27,hi8(taulac2+SALTA)
sigui2: ld r16,X+
    call tx
    cpi r26,lo8(taulac2_fin)
    brne sigui2
    reti

main:
    /* set baud rate a 9600*/
    ldi r16, 0
    sts UBRR0H,r16
    ldi r16, 103
    sts UBRR0L,r16

    /* set frame format */
    /* el valor de reset ja és correcte:
    asíncron, sense paritat, 1 stop, 8 dades,
    velocitat normal, comunicació no multiprocessor */
    /*arreglem el bit U2X0 forçant el valor de reset
    al registre UCSR0A*/
    ldi r16, 0b00100000
    sts UCSR0A, r16

    /* enable rx, tx, amb interrupció d'rx */
    ldi  r16, 0b10011000
    sts UCSR0B,r16

    /*habilitem interrupcions */
    sei

    /*el bucle principal no fa res*/
loop: rjmp loop
    ret


.section .data

.include "p8-taula.s"

texto:
    .ascii "Hola que tal"
texto_fin:

.section .bss
vegades:
    .byte 0
bufer:
```

```
    .space 32,0 /*repeteix 32 vegades, 1byte, amb valor 0. Directiva .fill .skip també pot servir */

.global ___do_copy_data
.global ___do_clear_bss
```

La taula estableix l'equivalència del xifrat. Aquesta taula s'inclou en el codi amb la directiva d'assemblador *.include*. La primera columna defineix el text en clar i la segona columna defineix el text xifrat. Aquesta taula ha estat inspirada pels codis d'un teclat PS2, modificant alguns d'aquests codis per evitar els codis no imprimibles.

```
/* Definició de la taula de xifrat */

taulac1:
    .byte 0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4A,0x4B,0x4C,0x4D,0x4E,0x4F,0x50,0x51,0x52,0x53
taulac1_fin:

taulac2:
    .byte 0x5C,0x32,0x21,0x23,0x24,0x2B,0x34,0x33,0x43,0x3B,0x42,0x4B,0x3A,0x31,0x44,0x4D,0x55,0x2D,0x5
taulac2_fin:
```

5 Estudi previ

TASCA PRÈVIA 1 Des-assembleu el codi executable *.elf* de *p8-exemple.s* usant:

```
avr-objdump -S a.elf > a.disam
```

Comproveu que el programa fa el que toca i descriviu els grans blocs que forma el des-assemblat del codi. Heu trobat on és la taula de xifrat?

TASCA PRÈVIA 2 Executeu

```
avr-objdump -s a.hex
```

i descriviu els grans blocs de dades que s'obtenen. Localitzeu la part de codi executable, dades inicialitzades i dades no inicialitzades (inicialitzades a 0). Heu trobat totes les seccions?

TASCA PRÈVIA 3 Dissenyeu una programa per tal que quan es rebi el caràcter '1' es retorni el contingut de la posició SALTA de la primera columna, i quan es rebi el caràcter '2' es retorni el contingut de la posició SALTA de la segona columna.

TASCA PRÈVIA 4 Modifiqueu el programa del previ 3 per tal que quan DEBUGAR valgui 1, per cada byte rebut es retorni primer el caràcter rebut i després el caràcter corresponent al previ 3.

TASCA PRÈVIA 5 Modifiqueu el programa del previ 4 per tal que la variable de 8 bits VEGADES contingui el nombre de caràcters rebuts des del moment de *reset*. Si DEBUGAR val 1, VEGADES serà el tercer caràcter retornat. Si DEBUGAR val 0, només es retornarà el byte corresponent al previ 3.

TASCA PRÈVIA 6 Dissenyeu un programa que busqui en la primera columna de la taula el caràcter rebut. Si la cerca és positiva, es retorna 'S' i, si la cerca és negativa, es retorna 'N'.

TASCA PRÈVIA 7 Dissenyeu el programa xifrador. Recordeu que com a caràcter rebut admet 'A' a 'Z', 'a' a 'z' i l'espai. Les majúscules i minúscules es tracten per igual, i el caràcter espai no s'ha de xifrar. Si es rep qualsevol altra caràcter no s'ha de retornar res.

TASCA PRÈVIA 8 *Opcional:* Dissenyeu el programa desxifrador basat en l'operació inversa al xifrador.

TASCA PRÈVIA 9 *Opcional:* Dissenyeu un programa xifrador/desxifrador. Si el primer caràcter rebut és '0' està en mode xifrador i si és '9' en mode desxifrador.

Podeu modificar el vostre codi del previs 4, 5, 6 i 7 per tal de contemplar un mode de DEBUGAR al vostre gust.

6 Treball pràctic

TASCA 10 **El treball al laboratori** consisteix en la comprovació de les tasques de l'estudi previ.

A continuació teniu algunes comandes del *Toolchain* de GNU, imprescindibles per passar els codis al microcontrolador.

```
avr-gcc -mmcu=atmega328p -o a.elf a.s

avr-objcopy --output-target=ihex a.elf a.hex

avrdude -c arduino -P /dev/ttyACM0 -p m328p -U flash:w:a.hex:i
```

Referències

[AS] Manual de referència del Assemblador del GNU - `http://sourceware.org/binutils/docs/as/`