UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Enginyeria de Sistemes TIC

Generador de Polsos

Dispositius Programables — Enginyeria de Sistemes TIC

Jordi Bonet Francisco del Águila

6 de novembre de 2022

Índex

1 Objectiu

L'objectiu d'aquesta tercera sessió és aprofundir en la utilització de l'assemblador amb la introducció de les crides a subrutines i les macro, i continuar usant els ports d'entrada / sortida del microcontrolador.

2 Introducció teòrica

2.1 Crida a subrutines

En general, tots els microcontroladors tenen instruccions de crida a subrutina. En el cas dels AVR són **call**, **rcall** i **icall** amb la seva corresponen instrucció de retorn **ret**. Una crida a una subrutina es comporta similar a les instruccions de salt incondicional **jmp** i **rjmp** però amb la peculiaritat que una vegada s'ha executat un tros de codi concret i s'arriba a la instrucció **ret** l'execució del programa torna a la posició de memòria següent a la d'on s'ha fet la crida, és a dir a la instrucció posterior a **call**, **rcall** o **icall**. Això és útil per executar un tros de codi concret que pot apareix en més d'un lloc del programa principal. Amb la crida a subrutines, el codi només apareix en un lloc del programa principal, amb l'avantatge que a) si el codi es modifica només s'ha de modificar en un lloc i b) s'usa menys memòria de programa. Una descripció més detallada d'aquestes instruccions les trobem a [Instr].

El funcionament d'aquestes crides, implica que el microcontrolador ha de recordar en quina posició de memòria de programa (Program Counter: **PC**) s'ha produït aquesta crida. De tal manera que un cop s'executa el codi de la subrutina i es troba la instrucció **ret** al final, es recupera la posició que tenia el **PC**. Aquesta posició de memòria és guarda a l'*stack*. Una descripció més detallada de l'*stack* la trobem a l'apartat 7.5 de [ATmega328p]. Com s'indica a la documentació, l'*stack* també es pot fer servir per guardar dades amb les instruccions **push** i **pop**.

2.2 Macros

Les macros serveixen per definir un tros de codi que l'assemblador anirà enganxant allà on es faci referència a la macro. Per tant és una tasca que realitza l'assemblador en el moment de crear el codi executable. Simplement, estalvia la feina al programador d'anar repetint el tros de codi en el moment de crear el codi font. Una descripció la trobem a la directiva **.macro** d'[AS].

Quan a dins d'una macro s'han de fer servir etiquetes, *labels*, a adreces de programa s'ha d'anar amb compte, ja que si s'usa la macro en més d'un lloc del programa hi haurà diferents posicions de programa amb la mateixa etiqueta, cosa que l'assemblador no sap interpretar. Una manera de solucionar aquest problema és fer servir etiquetes locals. Aquestes etiquetes es defineixen amb un nombre seguit de dos punts (xx:) i quan es fa referència a aquestes labels es pot afegir la lletra *b* o *f* per indicar a l'assemblador que és la primera etiqueta *backward* o *fordward* que trobi en el codi. Un exemple de funcionament es veu en el codi *segon.s*.

2.3 Ports d'entrada / sortida

Els AVR disposen de diferents ports d'entrada / sortida que estan mapejats a la memòria d'entrada / sortida, que a la vegada està solapada a la memòria de dades. Aquests ports estan connectats a les potes del circuit integrat. La configuració d'aquests ports es troba a l'apartat 14 d'[ATmega328p].

3 Exemples de programa

Els dos següents exemples de programa s'encarreguen de encendre i apagar el LED de l'Arduino a una freqüència determinada. Tots dos tenen el mateix efecte, tot i usar estructures diferents: un usa la crida a subrutina i l'altre usa una macro.

```
/* primer.s, usa la crida a subrutina */
.set DDRB_o , 0x4
.equ PORTB_o , 0x5

.global main

waitabit:
    ldi r19,41
wait3: ldi r18,0xFF
wait2: ldi r17,0xFF
wait1: subi r17,0x01
    brne wait1
    subi r18,0x01
    brne wait2
```

```
        subi r19,0x01
        brne wait3
        ret

main:   ldi r16,0xFF
        out DDRB_o,r16
loop:   ldi r16,0x00
        out PORTB_o,r16
        rcall waitabit
        ldi r16,0xFF
        out PORTB_o,r16
        rcall waitabit
        rjmp loop /* loop forever */

/* segon.s, usa una macro */
.set DDRB_o , 0x4
.equ PORTB_o , 0x5

.macro waitabit tot
        ldi r19,\tot
3:      ldi r18,0xFF
2:      ldi r17,0xFF
1:      subi r17,0x01
        brne 1b
        subi r18,0x01
        brne 2b
        subi r19,0x01
        brne 3b
.endm

.global main

main:   ldi r16,0xFF
        out DDRB_o,r16
loop:   ldi r16,0x00
        out PORTB_o,r16
        waitabit 41
        ldi r16,0xFF
        out PORTB_o,r16
        waitabit 41
        rjmp loop /* loop forever */
```

Observeu que a) en la macro s'usen etiquetes locals i b) la macro es defineix amb el paràmetre *tot*. Cada cop que s'usa la macro cal especificar el valor d'aquest paràmetre. Observeu que cada cop que s'usa la macro es pot usar un valor de paràmetre diferent, cosa que afegeix utilitat a la macro.

Noteu que tant les subrutines com les macros és poden veure com la definició d'una nova instrucció construïda amb les instruccions ja existents.

4 Estudi previ

TASCA PRÈVIA 1 Observeu que el tros de codi de la subrutina i la macro dels codis anteriors són idèntics. Calculeu el nombre de cicles màquina (*clk*) que es tarda en executar aquest tros de codi en funció del valor que es carrega al registre *r19* (*paràmetre*) en la primera instrucció.

TASCA PRÈVIA 2 Tenint en compte que el rellotge del microcontrolador és de 16 MHz, calculeu el valor que s'ha de carregar al registre *r19* per tal que el temps total que tarda en executar-se aquest tros de codi sigui de 0.5 s.

TASCA PRÈVIA 3 Una manera per crear un to audible amb una sortida digital és generant una ona quadrada de freqüència audible, per exemple $f_{to} = 600\,\text{Hz}$. Això ho podem aconseguir commutant entre 0 i 1 una de les potes de sortida de l'Arduino. Comproveu si la subrutina / macro *waitabit* serveix per poder generar una ona quadrada de $f_{to} = 600\,\text{Hz}$ en una de les potes de l'Arduino. Si ho considereu oportú, modifiqueu *waitabit*. Anomeneu a la nova subrutina / macro *waitato*.

TASCA PRÈVIA 4 Escriviu el codi assemblador que configura l'Arduino amb la pota 8 i 13 (LED) com a sortida i la pota 7 com a entrada. Aprofiteu els *pull-ups* interns, que es controlen amb els registres *DDRx*, *PORTx* i el bit *PUD* del registre *MCUCR*.

TASCA PRÈVIA 5 Escriviu en el fitxer *p3-codi1.s* un programa en assemblador que generi un to de $f_{to} = 600\,\text{Hz}$ en la pota 8 de l'Arduino.

TASCA PRÈVIA 6 En el fitxer *segon.s* s'ha definit una macro amb un paràmetre. Proposeu una manera d'aconseguir una funcionalitat similar (ús d'un paràmetre) quan des del programa principal del fitxer *primer.s* es fa una crida a la subrutina *waitabit*.

TASCA PRÈVIA 7 Escriviu en el fitxer *p3-codi2.s* un programa en assemblador que generi un to de $f_{to} = 600\,\text{Hz}$ en la pota 8 de l'Arduino, i que al mateix temps encengui i apagui el LED a una $f_{LED} = 2\,\text{Hz}$.

TASCA PRÈVIA 8 Escriviu en el fitxer *p3-codi3.s* un programa en assemblador per tal que quan a la pota 7 de l'Arduino hi hagi una tensió de 0 V encengui el LED i al cap d'1.5 s l'apagui. Un cop apagat, es torna a comprovar el valor de la pota 7, de manera que si encara hi ha una tensió de 0 V el LED es torna a encendre durant 1.5 s.

TASCA PRÈVIA 9 Modifiqueu el fitxer *p3-codi3.s*, i anomeneu-lo *p3-codi4.s*, per tal que quan a la pota 7 de l'Arduino hi hagi una tensió de 0 V es generi un to de $f_{to} = 600\,\text{Hz}$ durant 1.5 s en la pota 8 de l'Arduino.

5 Treball pràctic

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
```

5.1 Tasques

TASCA 10 Assembleu els fitxers *primer.s* i *segon.s*, transferiu-los a l'Arduino i comproveu que no es poden observar diferències en l'efecte que tenen sobre el LED. Des-assembleu els dos programes.

```
avr−objdump −S nom.elf > nom.disam
```

Descriviu les diferències entre els dos fitxers *.asm*.

TASCA 11 Comproveu el correcte funcionament de *p3-codi1.s*.

TASCA 12 Comproveu el correcte funcionament de *p3-codi2.s*.

TASCA 13 Comproveu el correcte funcionament de *p3-codi3.s*.

TASCA 14 Comproveu el correcte funcionament de *p3-codi4.s*.

Referències

[Ard] Arduino UNO; `http://arduino.cc/en/Main/ArduinoBoardUno`

[ATmega328p] Atmel. ATmega328P datasheet; `https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf`

[Instr] Joc d'instruccions dels AVR; `http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf`

[AS] Manual de referència de l'assemblador del GNU; `http://sourceware.org/binutils/docs/as/`