UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Enginyeria de Sistemes TIC

Primeres proves

Dispositius Programables — Enginyeria de Sistemes TIC

Jordi Bonet Francisco del Águila

19 de setembre de 2023

Índex

1 Objectiu

L'objectiu d'aquesta segona pràctica és intensificar el coneixement de les eines de treball i del *Toolchain* de GNU que ja vam usar en la primera pràctica per a injectar els primers programes escrits en assemblador en el microcontrolador de l'Arduino UNO.

2 Introducció teòrica

L'aplicació *avrdude* es fa servir per transferir els fitxers binaris cap a cadascuna de les possibles memòries del dispositiu. En general si el nostre fitxer binari prové de l'assemblat d'un programa i per tant és un codi binari executable, la memòria a la que transferirem aquest fitxer serà la memòria de programa del microcontrolador.

Les diferents memòries a les que podem accedir des de l'*avrdude* són:

flash: és la memòria en la que queda emmagatzemat el nostre programa, per tant és la memòria que típicament farem servir.

eeprom: és una memòria extra que tenen els AVR. A aquesta memòria no es pot accedir directament a través del bus d'adreces sinó que s'ha d'accedir-hi mitjançant accessos a registres d'entrada sortida. Típicament és on es pot guardar un gran volum de dades de forma no volàtil.

hfuse , lfuse , efuse , ... la resta de possibles memòries són petites zones reservades en els AVR per poder configurar paràmetres especials, per exemple l'identificador de dispositiu, certes configuracions, etc.

Les memòries a les que es pot accedir es poden consultar en el manual accessible executant *man avrdude* en un terminal de Linux. Cal buscar l'opció *-U*. En la descripció d'aquesta opció també es troben els formats de fitxers que admet l'avrdude.

Els formats dels fitxers poden ser:

r Format binari directe amb bytes ordenats de format little-endian [Endian].

i *Intel Hex* [ihex].

d Format de sortida (no es pot fer servir per escriure a la memòria, tan sols per llegir) decimal, separat per comes.

h Format de sortida hexadecimal. Cada valor va precedit de 0x.

b Format de sortida binari. Cada valor va precedit de 0b.

2.1 Els fitxers amb format ihex

El format *Intel Hex* és un format de text, i per tant visible directament, on cada línia conté valors hexadecimals que codifiquen les dades i la seva posició en forma d'adreça. A la referència [ihex] queda descrit aquest format.

2.1.1 Ordre de disposició dels Bytes

Per llegir o escriure totalment una memòria es segueix l'ordre determinat per les adreces, començant des de la posició zero fins al final. En el cas de memòries disposades en paraules d' 1 Byte (8 bits), això no comporta cap complicació afegida. Però en el cas de memòries amb amplada de paraula més gran que 1 Byte, per accedir a una paraula, mantenint la condició d'accedir a blocs de dades d'1 Byte, s'ha de pactar si l'accés es fa de dreta a esquerra o bé d'esquerra a dreta. Escollir una opció o l'altra és el que determina l'*Endiannes* de l'arquitectura de la memòria [Endian]. En l'arquitectura AVR aquest ordre és de tipus *Little Endian*. Això vol dir que quan volem omplir una memòria (ja sigui de dades o de programa) omplirem les posicions de dreta a esquerra. En el cas de la memòria de programa, les paraules són de 2 Bytes (16 bits). És per això que la matriu de memòria segueix l'ordre indicat per la següent taula:

1	0
3	2
5	4

2.2 El mapa de memòria dels AVR

L'arquitectura dels AVR és de tipus Harvard [Harvard]. La descripció detallada d'aquesta arquitectura i d'altres es veurà a les classes de teoria. Essencialment l'arquitectura Harvard es caracteritza per tenir separats, i per tant en busos diferents, els diferents tipus de memòria. L'existència en paral·lel de diferents busos permet que es puguin realitzar transferències simultànies a les diferents memòries. Per exemple, es pot llegir de la *memòria de programa* la següent instrucció a executar, al mateix temps que es pot llegir una dada de la *memòria de dades*.

En general hi ha tres tipus de memòries (a part de la memòria formada per un conjunt de registres de propòsit general):

Memòria de programa és l'espai reservat a contenir el codi executable. Està disposada en paraules de 16 bits (2 Bytes).

Memòria de dades aquí trobem l'espai disponible per emmagatzemar les dades. Està disposada en paraules de 8 bits (1 Byte).

Memòria d'entrada / sortida és l'espai on es troben els registres dels dispositius que configuren i controlen la comunicació i/o interacció amb l'exterior de la CPU. Per tant, aquestes registres gestionen els perifèrics del microcontrolador. Està disposada en paraules de 8 bits (1 Byte).

En el cas particular dels AVR la memòria d'entrada / sortida i la de dades aprofiten el mateix bus de dades. El bus de dades de la memòria de programa (allà on es carregarà la següent instrucció a executar) es troba totalment aïllat del bus de dades de la memòria de dades. Respecte al bus d'adreces de la memòria d'entrada / sortida cal dir que també es troba solapat sobre el bus d'adreces de la memòria de dades. Aquest solapament consisteix en que per adreçar-se als registres de la memòria d'entrada / sortida, a part de poder accedir amb les adreces pròpies del bus d'entrada / sortida, també es pot adreçar als mateixos registres amb adreces de memòria de dades, però ara la primera posició correspon a l'adreça 32, vegeu apartat *SRAM Data Memory* dins d''*AVR Memories* de [ATmega328p]). A més, les adreces 0 a 31 (les 32 primeres posicions) de la memòria de dades corresponen també a les adreces dels registres de propòsit general. Per tant, també existeix solapament sobre el mapa de memòria de dades dels registres de propòsit general.

El solapament de la memòria d'entrada / sortida implica que podem accedir a ella per dues vies diferents:

1. Una de les vies és fer servir les instruccions de codi màquina específiques per les entrades / sortides (**in**, **out**), però en aquest cas s'ha de tenir en compte que l'adreçament comença a comptar des de la posició 0.

2. L'altra via és fer servir les instruccions generals d'accés a memòria de dades (**ld**, **lds**, **st**, **sts**, ...) però en aquest cas hem de tenir en compte que la primera posició d'entrada / sortida és l'adreça 32.

Un llistat dels registres d'entrada / sortida es troba al capítol *Register Summary* de [ATmega328p].

2.2.1 Registres de propòsit general

Les 32 primeres posicions de la memòria de dades la formen uns registres especials. Aquests registres tenen el privilegi de poder ser tractats amb un conjunt elevat d'instruccions d'assemblador

que altres posicions del mapa de memòria no poden (veure apartat *General Purpose Register File* dins d'*AVR CPU Core* de [ATmega328p]).

2.3 Algunes de les instruccions en assemblador

Pel desenvolupament d'aquesta pràctica cal el coneixement previ d'algunes de les instruccions dels AVR. La descripció detallada del joc d'instruccions es troba a [Instr].

2.4 Algunes directives de l'assemblador de GNU

Les directives d'assemblador permeten definir aspectes característics de l'assemblador que estem fent servir. El conjunt format per les directives i el joc d'instruccions de cada CPU és el que forma un programa en assemblador preparat per poder ser assemblat.

Podem trobar la definició de les directives que es poden fer servir en el manual de l'assemblador del GNU, [AS]. Alguna d'aquestes directives són:

.global Necessita com a paràmetre un símbol. Fa que aquest símbol pugui ser visible per l'enllaçador o *linker*. A efectes pràctics, ara que estem a l'inici del curs, cal definir l'etiqueta *main:* que indica l'inici del programa i s'ha de definir com a *global*.

.equ , .set , = Aquestes tres directives són equivalents. Qualsevol de les tres permeten associar a un símbol una expressió qualsevol.

3 Recuperem el darrer programa

A continuació reproduïm el *tercer programa* de la primera sessió que, recordem, canvia alternativament l'estat de tots els pins del *PORTB*, entre ells el pin *PORTB5* que controla l'estat del LED taronja.

```
.global main

/* Aquí es fan algunes definicions fent servir tres directives equivalents */
   mregistre  = 16
   .set  DDRB_o , 0x4
   .equ  PINB_o , 0x3

/* Comença el programa principal */
main:
   ldi   mregistre,0xFF
   out   DDRB_o,mregistre
loop:
   out   PINB_o,mregistre
   rjmp  loop
```

S'observa la definició de l'etiqueta d'inici de programa *main:* i la utilització de la directiva **.global**. També observem les tres directives que permeten associar a un símbol una expressió qualsevol. Tot i que en aquest codi apareixen les tres directives, allò que és usual és utilitzar-ne un sol tipus. Observeu també que existeixen tres registres associats al set de potes *B*: *DDRB* que decideix si els pins són d'entrada o sortida; *PINB*, quan les potes estan configurades com a sortides, en el moment d'assignar un '1' a un dels seus bits, fa un *toggle* del pin corresponent';

PORTB que assigna, en mode sortida, directament un valor '0' o '1' als pins. Aquest darrer registre no apareix en aquest program, però sí en els anteriors. De manera simplificada, al set de potes *B* format per aquests 3 registres i la circuiteria electrònica associada, també se l'anomena *PORTB*. Per tant, cal diferenciar entre el *PORTB* genèric i el registre *PORTB*.

3.1 Diferents formats numèrics per enters

La manera de definir nombres enters en l'assemblador del GNU *as* es troba a l'apartat 3.6.2.1 de [AS].

El resum és:

	Prefix	**Exemple**
binari	0b	0b11001
octal	0	07430
decimal		1395
hexadecimal	0x	0xF30C91

4 Estudi previ

TASCA PRÈVIA 1 Del següent fitxer en format *Intel Hex* trobeu l'adreça on es troben les dades i extraieu en format hexadecimal els bytes que defineixen aquestes dades. No tingueu en compte el camp de *checksum*. Us caldrà consultar [Endian].

```
:04000000000000C0XX
:00000001FF
```

TASCA PRÈVIA 2 Considereu els bytes extrets del fitxer anterior com els *opcodes* d'instruccions màquina de l'AVR. Deduïu a quina de les següents instruccions màquina corresponen: **nop**, **rjmp**, **ldi**, **out**, **subi**, **brne**. Tingueu en compte que la disposició de memòria és del tipus *Little Endian*. Us caldrà consultar [Instr] i [Endian].

TASCA PRÈVIA 3 Considereu el codi assemblador

```
.global main

main:
    nop
    rjmp main
```

de la primera sessió. Extraieu els bits corresponents als opcodes d'aquestes instruccions i escriviu-los en un fitxer en format *Intel Hex* sense tenir en compte el checksum. Recordeu que cal usar la disposició *Little Endian*. Us caldrà consultar [Instr] i [Endian].

TASCA PRÈVIA 4 Tenint en compte que la memòria d'entrada / sortida està solapada a sobre de la memòria de dades, quin sentit pot tenir definir instruccions específiques d'entrada / sortida si amb les instruccions generals d'accés a la memòria de dades podem fer el mateix.

TASCA PRÈVIA 5 Busqueu a [Instr] què fan i quins paràmetres fan servir les instruccions de llenguatge assemblador: **nop**, **rjmp**, **ldi**, **out**, **subi**, **brne**.

TASCA PRÈVIA 6 Busqueu a [AS] quin significat tenen les directives **.global**, **.set**, **.equ**.

TASCA PRÈVIA 7 Localitzeu el pin de la plataforma Arduino UNO on està connectat el LED de color taronja. Podeu clicar sobre *Pinout Diagram* i *Interactive Viewer* de [Ard]. Identifiqueu quins registres del mapa d'entrada / sortida gestionen la configuració d'aquest pin de l'Arduino. Us caldrà consultar el capítol *Register Summary* de [ATmega328p]. Quin bit d'aquest registre controla el LED? Quina és l'adreça del registre d'entrada / sortida que permet la modificació de l'estat del LED (des del punt de vista del mapa d'entrada / sortida)? Quina és l'adreça del registre d'entrada / sortida que permet la modificació de l'estat del LED (des del punt de vista del mapa de memòria)?

TASCA PRÈVIA 8 Quines instruccions caldria usar per modificar l'estat del LED actuant sobe el mapa d'entrada / sortida?

TASCA PRÈVIA 9 Quines instruccions caldria usar per modificar l'estat del LED actuant sobe el mapa de memòria de dades?

TASCA PRÈVIA 10 Considerant la freqüència de *clk* de l'Arduino UNO i el nombre de cicles de *clk* que cal per executar les instruccions **ldi**, **out**, **rjmp**, calculeu a quina freqüència commutarà el LED del *segon programa* de la primera sessió, que reproduïm a continuació.

```
.global main

/* Això és un comentari
de més d'una línia  */

/* Aquí es fan algunes definicions fent servir dues directives equivalents */
mregistre = 16
.set DDRB_o , 0x4
.equ PORTB_o , 0x5

/* Comença el programa principal */
main:
    ldi   mregistre,0xFF
    out   DDRB_o,mregistre
loop:
    ldi   mregistre,0x00
    out   PORTB_o,mregistre
    ldi   mregistre,0xFF
    out   PORTB_o,mregistre
    rjmp  loop
```

Calculeu el temps en què el LED està en ON i en OFF.

TASCA PRÈVIA 11 Modifiqueu el codi del *segon programa* per tal que el temps en què el LED està en ON i en OFF sigui el mateix. No useu la propietat de *toggle*!!! Anomeneu a aquest quart fitxer *p1-codi4.s*.

TASCA PRÈVIA 12 Modifiqueu el codi del fitxer *p1-codi4.s* per tal de dividir per dos la freqüència de commutació del LED. Anomeneu a aquest cinquè fitxer *p1-codi5.s*.

TASCA PRÈVIA 13 **Tasca avançada:** Modifiqueu el codi del fitxer *p1-codi4.s* per tal que la freqüència de commutació del LED sigui visible a simple vista (inferior a 20 Hz). L'objectiu és entretenir el microcontrolador abans que faci el següent canvi d'estat del LED. Anomeneu a aquest sisè fitxer *p1-codi6.s*.

5 Treball pràctic

Al laboratori continuarem usant el *Toolchain* de GNU i comprovarem el funcionament dels arxius generats en l'estudi previ usant, si cal, l'oscil·loscopi.

5.1 Assemblat del codi font

Per assemblar el nostre codi font *nom.s* s'ha de cridar a la comanda

```
avr−gcc −mmcu=atmega328p −o nom.elf nom.s
```

on el paràmetre **-mmcu** indica el model de microcontrolador que estem fent servir, **-o** indica el fitxer final generat (format *elf*) i l'últim paràmetre és el fitxer amb el codi font. És a dir, *nom.s* és el fitxer d'entrada a l'assemblador i *nom.elf* és el fitxer generat de sortida.

Per convertir el format *elf* a *ihex*, la comanda és

```
avr−objcopy −−output−target=ihex nom.elf nom.hex
```

Una funcionalitat molt interessant és el procés de des-assemblat. En aquest cas, a partir d'un fitxer executable de tipus *elf* s'obté un fitxer en codi font. Òbviament la informació extra que conté el codi font original ha desaparegut. La comanda és

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

També és possible des-assemblar a partir del fitxer de tipus *hex*. En primer lloc s'ha de transformar el fitxer *hex* en un fitxer *elf* i posteriorment fer servir la comanda anterior substituint el paràmetre *-S* per *-D* ja que el nou fitxer *elf* que es crea ha perdut informació important utilitzada per des-assemblar. La comanda per convertir un fitxer *hex* en un fitxer *elf* és:

```
avr−objcopy −I ihex nom.hex −−output−target=elf32−avr nom.elf −B avr −v
```

5.2 Assemblat del codi font sense complements

El procés de l'apartat anterior és el procés normal que s'ha de fer quan es vol programar el microcontrolador AVR. Aquest procés implica que l'assemblador afegeix alguns extres en el codi per tal de deixar ben configurades altres funcionalitats de l'AVR com per exemple les interrupcions,

l'apuntador de la pila, la inicialització d'alguns registres, etc. Aquests extres provenen dels fitxers *startup* i *crt*.

Per evitar afegir en el codi executable aquesta inicialització i deixar estrictament el programa, la comanda és la següent

```
avr−gcc −mmcu=atmega328p −nostartfiles −o nom.elf nom.s
```

Les altres comandes per des-assemblar o convertir al format *ihex* segueixen sent vàlides.

Recordeu que el manual de qualsevol comanda es pot consultar executant en un terminal

```
man nomcomanda
```

Però les comandes de la forma *avr-comanda* tenen un manual que és genèric per a totes les arquitectures (sigui *avr* o no). Per tant, el manual de la comanda genèrica es crida sense el prefix *avr-*

```
man gcc
man objcopy
man objdump
```

5.3 Transferència d'un fitxer executable amb *Avrdude*

Quan es connecta l'Arduino UNO a l'ordinador pel port USB, s'ha de comprovar que s'ha detectat el port de comunicació i per tant està reconegut pel sistema. Això es pot comprovar amb l'ordre

```
dmesg
```

A les darreres línies que retorna la comanda hauria d'aparèixer un nou dispositiu amb identificació **ttyACM0**.

Per comprovar que l'Arduino UNO està operatiu i disposat a ser programat (injectar o exportar codi), la comanda és

```
avrdude −c arduino −P /dev/ttyACM0 −p m328p
```

on els paràmetres **-c**, **-P** i **-p** indiquen, respectivament, que el tipus de programador usat per *avrdude* és el propi Arduino UNO, que el port de comunicacions és el nou port USB **ttyACM0** detectat i que el dispositiu amb què es treballa és un ATmega328p. Aquests paràmetres es poden consultar en el manual:

```
man avrdude
```

Per fer les transferències dels fitxers binaris es fa servir l'opció **-U** (consulteu el manual). Els paràmetres d'aquesta opció són la memòria a la que s'accedeix, si es fa lectura o escriptura, el fitxer que es vol transferir i el format del fitxer.

Per exemple, per tal de llegir i guardar la memòria de programa de l'AVR en el fitxer *nom.hex* amb el format *Intel Hex* usaríem la comanda

```
avrdude −c arduino −P /dev/ttyACM0 −p m328p −U flash:r:nom.hex:i
```

Per exemple, per tal d'escriure a la memòria de programa de l'AVR el fitxer *nom.hex* amb el format *Intel Hex* usaríem la comanda

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

5.4 Tasques

TASCA 14 Assembla el codi

```
.global main

main:
    nop
    rjmp main
```

i genera el fitxer executable amb nom *primer.elf*. Escull les opcions d'assemblat adequades.

TASCA 15 Converteix el fitxer *primer.elf* a *primer.hex*.

TASCA 16 Visualitza el fitxer *primer.hex* i comprova si hi trobes allò que esperes. Localitza els opcodes que s'han fet servir indicant la seva localització en el fitxer.

TASCA 17 Si la tasca anterior no ha tingut resultat satisfactori, realitza el procés invers de des-assemblat del fitxer *primer.elf* cap a *primer.disam* i visualitza el seu contingut. És el mateix que el fitxer amb el codi font en assemblador? Comenteu el resultat.

TASCA 18 Repeteix els passos 1-4 amb les opcions d'assemblat sense complements.

TASCA 19 Modifica el fitxer font afegint un parell d'instruccions **nop** al començament amb les opcions d'assemblat sense complements. Re-anomena el nou fitxer amb *primer2.s* i comprova els canvis produïts en els fitxers *elf, hex* i *disam*. Són els que esperaves? Visualitza el contingut i comenta'l.

TASCA 20 Transfereix el contingut de *primer.hex* a la memòria de programa de l'Arduino. Podem comprovar si s'està executant correctament?

TASCA 21 Assembla el *segon programa* i genera el fitxer executable amb nom *segon*. Transfereix el programa a l'Arduino i comprova el resultat. Podem comprovar si s'està executant correctament? Mesura la freqüència a la que commuta el LED i comprova que encaixa amb els càlculs de l'estudi previ.

TASCA 22 Transfereix el programa *p1-codi5.s* a l'Arduino i comprova si fa allò que s'espera.

TASCA 23 **Tasca avançada:** Transfereix el programa *p1-codi6.s* a l'Arduino UNO i comprova si fa allò que s'espera.

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/`

[ihex] Format de fitxer Intel HEX; `http://en.wikipedia.org/wiki/Intel_HEX`

[Harvard] Arquitectura de tipus Harvard; `http://en.wikipedia.org/wiki/Harvard_architecture`

[Endian] Ordre de disposició dels bytes; `http://en.wikipedia.org/wiki/Endianness`