UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Enginyeria de Sistemes TIC

Calculadora

Dispositius Programables — Enginyeria de Sistemes TIC

Jordi Bonet Francisco del Águila

18 de desembre de 2022

Índex

1 Objectiu

L'AVR pot realitzar operacions de suma i resta. En aquestes operacions es suma/resta el valor contingut en dos registres i es guarda el resultat en un d'aquests registres. Així, **tant els operands com el resultat són dades d'1 byte**.

L'objectiu d'aquesta pràctica és ampliar les operacions de suma i resta **a dades de 2 bytes**. Per tal de fer aquestes operacions amb 2 bytes, usarem les operacions amb dades d'1 byte de l'AVR. El procediment serà similar al que nosaltres realitzem quan sumen nombres en codificació decimal: per tal de sumar dues dades de, per exemple, 4 dígits sumem digit a digit començant per la dreta. Observeu que, en aquest procediment, quan la suma de dos dígits és superior a 9 només conservem el dígit de menor pes i el de major pes el traslladem a la següent suma de dos dígits (que per tant serà de tres dígits). Aquest trasllat del valor en excés a la següent suma s'anomena *carry* quan treballem amb binari.

Abans de poder realitzar aquestes operacions amb dades de 2 bytes, caldrà implementar una estratègia per tal de

- donar-li a l'AVR els dos operands i l'operació (suma o resta) que es vol realitzar, i

- que l'AVR ens retorni el resultat de l'operació.

En el cas d'una calculadora com les que esteu acostumats a usar, la suma dels nombres decimals *74* i *9* la realitzeu introduint en un teclat els caràcters *'74+9='*. Allò que aneu escrivint ho visualitzeu en una pantalla. Després de prémer la tecla *'='* la calculadora us retorna el resultat *83*.

Nosaltres **escollirem interactuar amb l'AVR usant el port sèrie**, mitjançant un teclat per transmetre cap a l'AVR i visualitzant els resultats que ens retorna, per exemple, en un terminal. La major dificultat conceptual d'aquest pràctica es troba en comprendre que **una cosa és la *representació* d'una dada i l'altre el *valor* d'aquesta dada**. Per exemple, *'4A'*, *'74'*, *'112'* i *'01001010'* són representacions d'un mateix valor, el 74. Observeu que per diferenciar valor de representacions, les representacions les escrivim entre *'* [1]. Cadascuna de les representacions anteriors corresponen a una codificació amb una base diferent i una longitud diferent. La primera és hexadecimal de longitud 2, la segona decimal de longitud 2, la tercera octal de longitud 3, i la darrera binària de longitud 8. Per evitar ambigüitats entre representacions, usem un conveni consistent en precedir la representació amb algun caràcter extra. Així, indiquem la representació hexadecimal precedint-la amb *'0x'* (*'0x4A'*), la octal amb *'0'* (*'0112'*), la binaria amb *'b'* (*'b01001010'*), i la decimal no té cap caràcter extra (*'74'*). No hem d'oblidar que **la representació que usa l'AVR és la binària**[2] i que abans de fer les operacions suma/resta de l'AVR potser haurem de fer conversions entre representacions per tal d'obtenir la binària, que és amb la que es realitzen aquestes operacions. Al capdavall, **allò que es rep/transmet pel port sèrie és un registre que conté 8 bits**.

Ara ens queda decidir **amb quina representació transmetem/rebem les dades a/de l'AVR usant el port sèrie**. I aquí comencen les dificultats.

En primer lloc perquè tenim vàries possibles representacions, tot i que potser pensareu que la representació decimal és l'única que caldria considerar, com en la calculadora de l'exemple anterior. Si bé és cert que aquesta seria la representació que més facilitaria a un usuari usar la calculadora, té l'inconvenient que la conversió de decimal a binari no és immediata i, a més, la gestió del nombres decimals negatius necessita del signe menys, cosa que complica encara més la conversió de representacions.

En segon lloc perquè quan usem un teclat i una aplicació com *picocom* allò que transmetem pel port sèrie ho fem byte a byte usant la codificació ASCII. La codificació ASCII és una relació biunívoca entre un *valor* i un *caràcter*[3]. I aquí comença l'*embolica que fa fort*. **Cal diferenciar entre valor i caràcter**: ara cal tornar a dir que allò que es rep/transmet pel port sèrie és un registre que conté 8 bits. Només a tall d'exemple, si decidim enviar el valor *74* ho podem fer de vàries maneres.

- Si pensem en la representació hexadecimal, escrivint en el terminal el caràcter *'4'* seguit del caràcter *'A'*. L'AVR haurà d'interpretar que el caràcter *'4'*, que es rebrà en un registre del port sèrie de recepció com un byte amb els bits *'00110100'*, seguit del caràcter *'A'*, que es rebrà en un registre del port sèrie de recepció com un byte amb els bits *'00110100'*, és el valor *74*.

- Si pensem en la representació decimal, escrivint en el terminal el caràcter *'7'* seguit del caràcter *'4'*. L'AVR haurà d'interpretar que el caràcter *'7'*, que es rebrà en un registre del

[1] Òbviament, quan escrivim el valor hem d'usar una representació que, per pacte, serà la decimal.

[2] En un abús del llenguatge assumim que cada bit del registre equival a cadascun dels '0' o '1' de la representació binària.

[3] Consulteu amb *man ascii* aquesta relació entre *valor* i *caràcter*.

port sèrie de recepció com un byte amb els bits *'00110111'*, seguit del caràcter *'4'*, que es rebrà en un registre del port sèrie de recepció com un byte amb els bits *'00110100'*, és el valor *74*.

- Si pensem en la representació octal, escrivint en el terminal els caràcters *'1'*, *'1'* i *'2'*...
- Si pensem en la representació binària, escrivint en el terminal els caràcters *'0'*, *'1'*, *'0'*, *'0'*, *'1'*, *'0'*, *'1'* i *'0'*...
- Si pensem en la codificació ASCII, haurem d'enviar un caràcter amb una codificació que coincideixi amb el valor que volem enviar: el caràcter *'J'* (relacionat amb el valor *74*), que es rebrà en un registre del port sèrie de recepció com un byte amb els bits *'01001010'*[4].

Els exemples anteriors són conceptuals i no compleixen, per exemple, que les dades siguin de 2 bytes. Això i la resta de detalls, com la codificació de la operació suma o resta, valors negatius i l'entrega del resultat apareixeran en els següents apartats.

Podríem resumir l'explicació anterior dient:

- Volem implementar operacions amb dades de 2 bytes usant les operacions ja existents d'1 byte.
- L'intercanvi d'informació (operands, operació i resultat) es realitza pel port sèrie. Nosaltres pactem de quina forma representem/codifiquem aquesta informació, pensant (altre cop convé repetir) que allò que es rep/transmet pel port sèrie és un registre que conté 8 bits.

2 Introducció teòrica

Per poder poder realitzar operacions aritmètiques de 2 bytes, reservarem el parell de registres r1:r0 per al primer operand i el parell r3:r2 per al segon operand, on r1 i r3 són els bytes més significatius. El resultat de la suma o resta (r1:r0) amb (r3:r2) el deixarem a r1:r0.

De la mateixa manera que amb les instruccions màquina de la CPU, els operands poden codificar tant números sense signe com números amb signe: és el programador qui decideix si els registres contenen una codificació amb signe o sense. La principal diferència en el cas de fer servir una codificació o altre està en els flags **C** o **S** que es tindran en compte en el resultat final. Consulteu [Op].

S'ha de permetre tant la suma com la resta. Recordeu que el procediment per restar és exactament igual que el de sumar si es realitza el complement a 2 de l'operand que es resta. Recordeu també que en la codificació amb complement a 2, el bit més significatiu ens dona informació sobre el signe.

Un exemple del funcionament usant complement a 2 en una resta en format hexadecimal per a una aritmètica basada en 1 byte seria:

0x05 - 0x04 = 0x01

que és equivalent a

0x05 -0x04 = 0x05 + (0xFB + 0x01) = 0x05 + 0xFC = 0x01

El programa s'ha d'estructurar com dues crides a subrutines, la de suma i la de resta, on els paràmetres són els registres r1:r0 i r3:r2.

Per donar a l'AVR els operands es farà ús del port serie (podeu fer servir les interrupcions o no). En primera opció (*entrada de dades directa*) es donaran directament els bytes que formen els

[4]Veurem que la utilitat *cutecom* té avantatges sobre *picocom* si volem usar aquesta opció o una de similar.

operands aprofitant l'aplicació **cutecom**. En segona opció (*entrada de dades amb representació hexadecimal*) es donarà la representació en hexadecimal dels operands, per tant, per cada operand s'entrarà un màxim de 4 caràcters. L'ordre en el que es donen els bytes, o parts de bytes, determinaran l'endianness del sistema de comunicació AVR - ordinador.

2.1 Representacions de nombres enters

Per poder representar un nombre enter existeixen diferents representacions: decimal, hexadecimal, octal, binària...

Per exemple, el nombre de valor *129* té les següents representacions:

- *'81'*: Hexadecimal, de longitud 2 caràcters. Cada caràcter forma part d'un grup de 16 dígits: '0' al '9' i 'A' a 'F'.
- *'129*: Decimal, de longitud 3 caràcters. Cada caràcter forma part d'un grup de 10 dígits: '0' al '9'.
- *'201*: Octal, de longitud 3 caràcters. Cada caràcter forma part d'un grup de 8 dígits: '0' al '7'.
- *'10000001'*: Binària, de longitud 8 caràcters. Cada caràcter forma part d'un grup de 2 dígits: '0' i '1'.

En el nostre cas, i per simplificar el problema, farem servir la representació hexadecimal en l'apartat *entrada de dades amb representació hexadecimal*. Per tant, per cada byte que vulguem transmetre, enviarem els 2 caràcters de la seva representació hexadecimal. Cadascun d'aquests caràcters s'enviarà en forma de byte amb el valor de la seva codificació ASCII. El receptor, en aquest cas l'AVR, ha de convertir aquests 2 bytes en 1 sol byte que contingui el valor que representen. Observeu que si al valor de la codificació ASCII dels caràcters '0' al '9' li restem el valor de la codificació del caràcter '0' s'obté el valor numèric que representen aquests caràcters. Una cosa similar es pot observar per als caràcters 'A' a 'F'.

2.1.1 Entrada de dades directa

La recepció de les dades es basa en la màquina d'estats de la figura 1 que, habitualment, segueix la següent seqüència de funcionament cada vegada que arrenca el programa o es fa un reset amb el polsador:

1. S'envia el primer byte del primer operand. L'AVR el retorna.
2. S'envia el segon byte del primer operant. L'AVR el retorna.
3. S'envia *'+' (0x2B)* o *'-' (0x2D)*. L'AVR el retorna.
4. S'envia el primer byte del segon operand. L'AVR el retorna.
5. S'envia el segon byte del segon operant. L'AVR el retorna.
6. S'envia *'=' (0x3D)*. L'AVR retorna el resultat enviant primer el primer byte i després el segon. La màquina d'estats retorna a l'estat inicial: esperar el primer byte del primer operand.

L'aplicació que permet enviar directament bytes pel port sèrie és **cutecom**, ja que amb **picocom** no és còmode enviar bytes corresponents a caràcters no imprimibles.

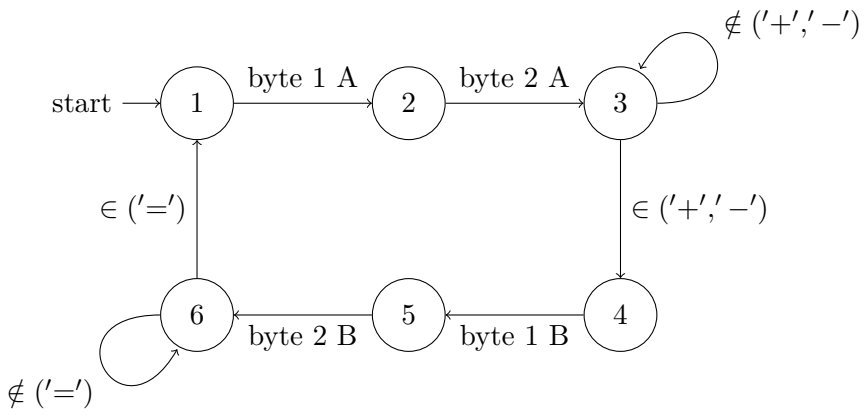

Figura 1: Màquina d'estats de l'entrada de dades directa.

2.1.2 Entrada de dades amb representació hexadecimal

La recepció de les dades es farà basada en una màquina d'estats, que vosaltres haureu d'implementar, seguint, habitualment, la següent seqüència de funcionament cada vegada que arrenca el programa o es fa un reset amb el polsador:

1. S'envien com a màxim els 4 caràcters de la representació hexadecimal del primer operand. L'AVR els retorna. En cas que se n'enviïn menys de 4, s'ha de considerar que els que falten són els de major pes i són '0'. L'ordre vindrà determinat per l'endianness escollit.

2. S'envia *'+'* o *'-'*. L'AVR el retorna.

3. S'envien com a màxim els 4 caràcters de la representació hexadecimal del segon operand. L'AVR els retorna. En cas que se n'enviïn menys de 4, s'ha de considerar que els que falten són els de major pes i són '0'. L'ordre vindrà determinat per l'endianness escollit.

4. S'envia *'='*. L'AVR retorna el resultat de l'operació enviant els 4 caràcters de la seva representació hexadecimal. La màquina d'estats retorna a l'estat inicial.

5. Si en qualsevol moment s'envia *'R'* pel port serie, la màquina d'estats retorna a l'estat inicial. L'AVR retorna *'R'* per indicar aquest fet.

2.1.3 Endianness de l'entrada de dades

El *llenguatge humà* fa servir el format Big Endian: quan llegim o escrivim la representació d'un nombre comencem pel caràcter de més pes. En el format Little Endian comencem pel caràcter de menys pes.

En les vostres implementacions escolliu el format que preferiu. Observeu que en el primer cas, entrada de dades directa, la vostra unitat d'informació és un caràcter que equival a 1 byte dels 2 que formen un operand, mentre que en el segon cas, entrada de dades amb representació hexadecimal, la vostra unitat d'informació és un caràcter que equival a 1 *nibble* dels 4 que formen un operand.

2.2 Programari de comunicació

2.2.1 Entrada de dades directe

En aquesta ocasió és necessari disposar d'una aplicació que permeti enviar bytes directament amb un valor determinat. Una aplicació que permet fer això és **cutecom**. Aquesta aplicació permet visualitzar les comunicacions entre l'AVR i l'ordinador tant en format ASCII com en format hexadecimal. A més, ens permet transmetre qualsevol byte corresponent o no a un caràcter imprimible.

Per instal · lar aquesta aplicació en una distribució tipus Debian s'ha d'executar la comanda

```
sudo aptitude install cutecom
```

Un dels detalls més importants per fer-la servir correctament, a part de configurar-la adequadament aprofitant els desplegables de la part superior, és el desplegable de la part inferior amb les opcions:

LF line end: afegeix el caràcter *(0x0A)* al final del text escrit al camp *input* quan polsem la tecla *intro* per enviar. El codi ASCII corresponent a la tecla *intro* no s'envia.

CR line end: afegeix el caràcter *(0x0D)* al final del text escrit al camp *input* quan polsem la tecla *intro* per enviar. El codi ASCII corresponent a la tecla *intro* no s'envia.

CR LF line end: afegeix els caràcters *(0x0D)* i *(0x0A)* al final del text escrit al camp *input* quan polsem la tecla *intro* per enviar. El codi ASCII corresponent a la tecla *intro* no s'envia.

No line end: No afegeix res quan polsem la tecla *intro* per enviar.

Hex input: No afegeix res quan polsem la tecla *intro* per enviar. S'envien els bytes amb el valor corresponent a la representació hexadecimal escrit al camp *input*.

2.2.2 Entrada de dades amb representació hexadecimal

L'ordinador que es connecta amb l'Arduino necessita d'un terminal que permeti la interacció entre l'usuari i l'AVR. La funció d'aquest terminal és permetre que tot el que l'usuari tecleja sigui transmès al dispositiu connectat i també permet que allò que el dispositiu envia cap a l'ordinador pugui ser visualitzat per la pantalla. Una eina d'aquest tipus és **picocom**.

Per instal · lar aquesta aplicació en una distribució tipus Debian s'ha d'executar la comanda

```
sudo aptitude install picocom
```

Per saber com funciona aquesta aplicació es pot fer ús del seu manual: *man picocom*. La comanda mínima per executar-lo és

```
picocom /dev/dispositiu_serie
```

En aquest cas agafa els valors de configuració per defecte (9600 bps, 8 bits de dades, sense paritat, 1 bit de stop...).

Amb aquesta aplicació, cada caràcter que l'usuari prem usant el teclat, es transmet pel port sèrie usant la codificació ASCII. De la mateixa manera, tot el que es rep pel port sèrie es descodifica usant la codificació ASCII i es mostra el caràcter corresponent per pantalla.

3 Estructuració del programa

En aquesta pràctica s'ha d'utilitzar algun codi que ja s'ha implementat en pràctiques anteriors:

- Utilització del mòdul USART.
- Implementació d'una màquina d'estats.
- Crida a subrutines.

Hi ha dues tasques importants, que funcionen de forma independent una de l'altra, per implementar:

1. La implementació de les dues rutines **suma** i **resta** que fan el càlcul amb precisió de 2 bytes. Per fer aquest càlcul, el flag de carry és de vital importància. Aquestes dues rutines han de ser totalment transparents.
2. La màquina d'estats que recull la informació d'usuari pel port serie, processa el bytes rebuts quan cal, i col · loca el valor dels operands en els registres adequats. En el cas de l'entrada de dades directe, la màquina d'estats és molt simple, figura 1. En el cas de l'entrada de dades amb representació hexadecimal, la màquina d'estats és més elaborada ja que ha de contemplar el reset per teclat *'R'* i l'entrada de dades incompleta (menys de 4 bytes).

4 Estudi previ

TASCA PRÈVIA 1 Dissenyeu la rutina **suma** que sumi el contingut dels registres r1:r0 amb r3:r2 i deixi el resultat a r1:r0.

TASCA PRÈVIA 2 Dissenyeu la rutina **resta** que resti el contingut dels registres r1:r0 amb r3:r2 i deixi el resultat a r1:r0. Feu-ne dues implementacions:

1. Implementeu-la usant la rutina **suma** anterior.
2. Implementeu-la sense usar la rutina **suma** anterior.

TASCA PRÈVIA 3 Dissenyeu la rutina **asc2val** que transformi el valor de la codificació ASCII dels caràcters '0' al '9' i 'A a 'F' en el valor d'aquests caràcters en una representació hexadecimal. Aquesta rutina agafa el caràcter del registre r20 i torna el resultat en el mateix registre. Per exemple: el valor 65, que és la codificació ASCII del caràcter 'A', s'ha de transformar en el valor 10, que és el valor de la representació hexadecimal 'A'.

TASCA PRÈVIA 4 Dissenyeu la rutina **val2asc** que faci el contrari que **asc2val**. Per exemple: el valor 10, que és el valor de la representació hexadecimal 'A', s'ha de transformar en el valor 65, que és la codificació ASCII del caràcter 'A'. Quin rang de valors d'entrada són vàlids pel registre r20?

TASCA PRÈVIA 5 *Entrada de dades directe.* Dissenyeu la màquina d'estats de la figura 1 amb l'endianness escollit.

TASCA PRÈVIA 6 *Entrada de dades amb representació hexadecimal.* Dissenyeu una màquina d'estats amb l'endianness escollit. S'ha de contemplar l'entrada del caràcter *'R'* per reiniciar la màquina d'estats i l'entrada parcial dels 4 caràcters que formen un operand. Abans de realitzar la implementació dibuixeu el seu diagram d'estats.

TASCA PRÈVIA 7 Justifiqueu l'endianness escollit en cadascuna de les dues implementacions anteriors.

TASCA PRÈVIA 8 Avançat: En el moment de donar el resultat amplieu la sortida amb 2 bytes més al principi, on el primer byte tindrà valor '0' o '1' en funció del flag **C** global de l'operació (en cas d'operands sense signe) i el segon byte tindrà valor '0' o '1' en funció del flag **S** global de l'operació (en cas d'operands amb signe).

5 Treball pràctic

TASCA 9 **El treball al laboratori** consisteix en la comprovació de les tasques de l'estudi previ **durant dues sessions**.

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

[Op] Operacions binàries amb signe i sense signe; `https://ocwitic.epsem.upc.edu/assignatures/dp/teoria/operacions-binaries-amb-signe-i-sense-signe/@@download/file/operacions_suma_resta.pdf`

[ASCII] Taula de caràcters ASCII; `http://en.wikipedia.org/wiki/ASCII`

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