Enginyeria de Sistemes TIC

Dispositius Programables

Pràctica 7 - Operacions aritmètiques

Francisco del Águila López

Novembre 2012
Escola Politècnica Superior d'Enginyeria de Manresa
Universitat politècnica de Catalunya

1 Objectiu

L'objectiu d'aquesta pràctica és ampliar l'operació aritmètica de suma i resta amb dades de 2 Bytes de precisió. Paral·lelament s'ha d'interactuar amb l'AVR per donar els números que s'han de sumar o restar, per tant s'ha d'implementar un mecanisme per adaptar les dades que dona l'usuari (codi ASCII) i convertir-les a codi binari de 2 Bytes.

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 o números amb signe i això dependrà del que consideri el programador. 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.

S'ha de permetre tant la suma com la resta. Recordeu que el procediment per restar és exactament igual que el de sumar però s'ha de realitzar el complement a 2 del operand que es resta. Recordeu també que en codificació en complement a 2, el bit més significatiu es considera el signe.

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

0x05 - 0x04 = 0x01

que és equivalent a

0x05 -0x04 = 0x05 + 0xFC = 0x01

El programa s'ha d'estructurar com dues crides a subrutines, la se 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 a través del port serie (podeu fer servir les interrupcions o no). En primera opció es donaran directament els bytes que formen els operands aprofitant l'aplicació **cutecom** (veure apartat 2.4). En segona opció 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. Per tant, si es dóna primer el byte de major pes del operand és format Big Endian. En canvi, si es dóna primer el byte de menor pes és format Little Endian.

2.1 Formats de codificació

Per evitar la utilització d'una aplicació especial, **cutecom**, que permeti l'enviament d'un byte amb la possibilitat d'escollir qualsevol valor, es farà servir un terminal genèric, **picocom**. L'inconvenient de fer servir un terminal es que la representació de qualsevol número es fa amb caràcters de text (codi ASCII). Per poder representar un número existeixen diferents representacions: binari, octal, decimal, hexadecimal.

Per exemple, el byte corresponent al valor 129 (decimal) té les següents representacions:

- 10000001: Binaria, són 8 lletres fent servir només els caràcters 1 i 0 de la taula ASCII.
- 201: Octal, són 3 lletres fent servir només els caràcters 0..7 de la taula ASCII. Valor més gran 377.
- 129: Decimal, són 3 lletres fent servir només els caràcters 0..9 de la taula ASCII. Valor més gran 255.
- 81: Hexadecimal, són 2 lletres fent servir només els caràcters 0..9 i les lletres A..F de la taula ASCII. Valor més gran FF.

En el nostre cas, per simplificar el problema, farem servir la representació hexadecimal. Per tant, per cada byte que volem transmetre, enviarem 2 caràcters (2 bytes) segons la codificació de la taula ASCII. El receptor, en aquest cas l'AVR, ha de convertir aquests 2 caràcters en el byte que representen. Com a suggeriment: la resta d'una constant al valor de la taula ASCII dóna el valor del caràcter que representa.

2.2 Entrada de dades directa.

La recepció de les dades es farà basada en una màquina d'estats seguint 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 el primer byte i el segon byte. 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 pot ser **cutecom**, ja que amb el **picocom** no és fàcil enviar bytes que no encaixin amb caràcters no imprimibles.

En aquest cas no hi ha possibilitat de fer un reset enviant *'r'* pel port sèrie, ja que si es donen directament els bytes no hi ha manera de distingir una *'r'* d'un byte qualsevol de l'operand.

2.3 Entrada de dades amb representació hexadecimal.

La recepció de les dades es farà basada en una màquina d'estats seguint 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 el retorna. En cas que s'enviïn menys de 4, s'ha de considerar que els que falten 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 el retorna. En cas que s'enviïn menys de 4, s'ha de considerar que els que falten són 0. L'ordre vindrà determinat per l'endianness escollit.
4. S'envia = l'AVR retorna el resultat enviant la representació hexadecimal del número. En aquest cas s'escriuran els 4 caràcters que el formen. La màquina d'estats retorna a l'estat inicial.
5. Si en qualsevol moment intermig s'envia *'r'* pel port serie, s'inicialitza la màquina d'estats i es torna a esperar el primer operand. L'AVR retorna *'r'* per indicar aquest fet.

Es recorda que s'ha de permetre que amb aquesta màquina quan s'introdueixin els operands en format hexadecimal, es pugui fer de manera parcial. Es a dir, si només s'envia una part de l'operand, s'ha d'entendre que la part que falta és 0. Així és com funcionen les calculadores que fem servir contínuament.

El llenguatge humà fa servir el format Big Endian. Quan llegim un número, comencem pel caràcter de més pes fins arribar a donar en última posició (Big endian) el de menys pes. El format Little endian comença donant les primeres posicions els caràcters de menys pes (Little endian) i posteriorment els de més pes. Per tant, escolliu la implementació que vulgueu per aquesta pràctica:

- Enviar els dos bytes de l'operand: el byte de més pes primer i el de menys pes després -> Big endian
- Enviar els dos bytes de l'operand: el byte de menys pes primer i el de més pes després -> Little endian

2.4 Programari de comunicació

L'ordinador que es connecta amb l'Arduino necessita d'un terminal que permeti la interacció entre l'usuari i el dispositiu amb el que es comunica. 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 podria ser **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, ...)

En aquesta ocasió és interessant no disposar d'un terminal com és el picocom, sinó d'una aplicació que permeti enviar bytes directament amb un valor arbitrari. 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 de poder transmetre qualsevol byte arbitrari.

Per instal·lar-la s'ha de fer

```
sudo aptitude install cutecom
```

2.5 Codificació ASCII

Un estàndard molt simple per poder codificar en binari els caràcters d'un text és la codificació ASCII [ASCII]. Aquesta codificació en el seu format més simple consisteix en

una taula de 7 bits. Amb 7 bits només es poden codificar 128 possibles valors per tant el conjunt de possibles caràcters és bastant reduït. Ja que la majoria de sistemes digitals treballen amb unitats de 8 bits (1Byte), la taula ASCII també està definida amb 8 bits. Això augmenta el nombre de caràcters al doble (256 valors), quedant definida la taula ASCII estesa.

La codificació binària que es fa servir per transmetre un caràcter a través d'una comunicació basada en un dispositiu USART és la codificació ASCII. Per exemple, la transmissió de la lletra **A** queda codificada amb els bits **0x41**.

3 Estructuració del programa

En aquesta pràctica que té una durada de dues sessions, s'ha de fer servir codi que ja s'ha fet servir en pràctiques anteriors:

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

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

1. La màquina d'estats que recull la informació d'usuari pel port serie i col·loca els bytes transmesos en els registres adequats. En el cas de la introducció de dades directament en bytes, la màquina d'estats és molt simple. En el cas de la introducció amb la representació hexadecimal, la màquina d'estats és més elaborada ja que ha de contemplar el reset per teclat 'r' i la introducció de dades incompleta.
2. La implementació de les dues rutines **suma** i **resta** que fan el calcul amb precisió de 2 bytes. Aquestes rutines s'han d'implementar de manera totalment independent de la màquina d'estats. Per fer aquest càlcul, el flag de carry és de vital importància.

4 Estudi previ

1. Dissenya una rutina **suma** que sumi el contingut de r1:r0 amb r3:r2 i deixi el resultat a r1:r0.
2. Dissenya una rutina **resta** que resti el contingut de r1:r0 amb r3:r2 i deixi el resultat a r1:r0. Implementa-la fent servir la rutina suma i implementa-la fent servir les instruccions màquina bàsiques.
3. Dissenya una rutina **asc2val** que transformi qualsevol caràcter 0..9 i lletra A..F en el seu valor corresponent. Aquesta rutina agafa el caràcter del registre r20 i torna el resultat al mateix registre.

4. Dissenya una rutina **val2asc** que faci el contrari que **asc2val**. Quin rang de valors d'entrada són vàlids pel registre r20?
5. Dissenya una màquina d'estats que reculli els dos operands i l'operació amb l'endianness escollida. Els bytes dels operands es transmeten directament sense fer servir cap representació. En aquest cas no s'ha de contemplar la possibilitat de introduir una r o bé permetre al introducció parcial dels operands. Abans de realitzar la implementació dibuixa el graf corresponent.
6. Dissenya una màquina d'estats que reculli els dos operands i l'operació amb l'endianness escollida. Els bytes dels operands es transmeten en representació hexadecimal. En aquest cas s'ha de permetre la introducció de la r i dels operands de manera parcial. Abans de realitzar la implementació dibuixa el graf corresponent.
7. Amplia la precisió a 8 bytes. Això suposa fer servir els registres r7:r0 pel primer operand i els r8:15 pel segon operand. Cal canviar la màquina d'estats?
8. Justifica avantatges i inconvenients de fer servir Big Endian o Little Endian.
9. Opcional: 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 carry global de la operació (en cas d'operands sense signe) i el segon byte tindrà valor '0' o '1' en funció del flag S global de la operació (en cas d'operands amb signe).

5 Treball pràctic

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

5.1 Assemblat del codi font

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

```
avr-gcc -mmcu=atmega328p -o prova.elf prova.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 directament el fitxer amb el codi font.

Un paràmetre que pot ser d'utilitat és **-E**, en aquest cas es realitza l'assemblat estrictament. Pot ser d'utilitat per analitzar possibles errors.

```
avr-gcc -mmcu=atmega328p -o prova.asm -E prova.S
```

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

```
avr-objcopy --output-target=ihex prova.elf prova.hex
```

Una possibilitat molt important 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 prova.elf > prova.disam
```

5.2 Transferència dels fitxers executables (Avrdude)

Quan es connecta l'Arduino al 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 línies finals hauria d'aparèixer un nou dispositiu amb identificació **ttyACM0.**

Per comprovar que l'Arduino està operatiu i disposat a acceptar ordres, la comanda és

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

on els paràmetres indiquen que el tipus de programador que fa servir l'*avrdude* és el que inclou directament el kit Arduino, que el port de comunicacions és el nou port USB detectat i que el dispositiu amb el que el 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, el format del fitxer. Un exemple per llegir la memòria de programa i crear un fitxer Intel Hex amb el seu contingut seria

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

Referències

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

[ATmega328p] Atmel. ATmega328P datasheet. `http://www.atmel.com/dyn/resources/prod_documents/doc8271.pdf`

[AtmAss] Atmel Assembler documentation - `http://www.atmel.com/atmel/acrobat/doc1022.pdf`

[Instr] Joc d'instruccions dels AVR - `http://www.atmel.com/atmel/acrobat/doc0856.pdf`

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

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

[Hardvard] Arquitectura de tipus Hardvard - http://en.wikipedia.org/wiki/Harvard_architecture

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

[USART] Dispositiu USART http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter

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