AGH
Akademia
Górniczo-Hutnicza
w Krakowie
Katedra Elektroniki

Technika mikroprocesorowa

Laboratorium 5

Operacje arytmetyczne

Autor: Paweł Russek
Tłumaczenie: Marcin Pietroń i Ernest Jamro
http://www.fpga.agh.edu.pl/tm
ver. 12.05.16

1. Wstęp

1.1. Cel ćwiczenia

Głównym celem laboratorium jest dokładniejsze zapoznanie się z operacjami arytmetycznymi procesora. W pierwszej części omówione zostanie reprezentacja liczb całkowitych ze znakiem. Przedstawiony zostanie problem przepełnienia oraz instrukcje arytmetyczne bazujące na rejestrze statusowym.

1.2 Wymaganie wstępne

•Rozumienie operacji arytmetycznych na liczbach ze znakiem.

•Zaliczenie ćwiczeń laboratoryjnych 1, 2, 3, oraz 4.

2. Przygotowanie płytki ZL2AVR

Użyj portu A do sterowania segmentami wyświetlacza alfanumerycznego LED. Podłącz wybrane końcówki portu B do anod wyświetlacza 7 segmentowego.

3. Liczby ze znakiem

Wartości 8-bitowe ze znakiem zapisywane są w kodzie uzupełnień do dwóch, gdzie najstarszy bit b7 (MSB – Most Significant Bit) ma ujemną wagę (-128 dla wartości 8-bitowej), natomiast bity b0 do b6 mają wagi dodatnie, tak jak w naturalnym kodzie binarnym. Jeżeli b7=0, wartość liczby jest dodatnia, jeśli b7=1, to jest ujemna. **Flaga N (Negative) w rejestrze statusowym stanowi kopię bitu b7.**

W przypadku wartości ujemnych bit b7 jest równy 1, a właściwa wartość reprezentowana jest w kodzie uzupełnień do dwóch (U2). Jeżeli w programie assembler zostanie podana wartość ujemna (na przykład: ldi r16,-7), assembler automatycznie dokonuje konwersji wartości na kod U2. Warto pamiętać, że liczby dodatnie w kodzie U2 mają reprezentację identyczną jak w naturalnym kodzie binarnym, ale b7 musi być zero. Zakres liczb dodatnich wynosi więc 0 do 127 (0b01111111). Zakres liczb ujemnych to -128 do 0.

Algorytm generacji liczby przeciwnej do danej w kodzie uzupełnień do dwóch jest następujący:

1. Napisz reprezentację binarną liczby.

2. Dokonaj zmiany każdego bitu na przeciwny.

3. Dodaj jeden do otrzymanej wartości.

Przykład:

Jak AVR reprezentuje liczbę -9?

1. *00001001 9 in 8-bit binary*
2. *11110110 invert bits*
3. *11110111 add 1 (number becomes $F7 in hex)*

Uwaga:

Dodatnie wartości ze znakiem są reprezentowane analogicznie jak wartości bez znaku. Przedział wszystkich możliwych wartości 8-bitowych typu całkowitego ze znakiem zawiera się w wartościach z zakresu od -128 do 127 w przeciwieństwie do liczb bez znaku o zakresie od 0 do 255.

Operacje arytmetyczne dla wartości ze znakiem i bez znaku są wykonywane przez identyczne instrukcje assemblera. Te same operacje procesora używa się dla obu reprezentacji.

Decimal	**Binary U2**	**Hex**
-128	1000 0000	0x80
-127	1000 0001	0x81
-126	1000 0010	0x82
…	…	…
-2	1111 1110	0xFE
-1	1111 1111	0xFF
0	0000 0000	0x00
1	0000 0001	0x01
2	0000 0010	0x02
…	…	…
127	0111 1111	0x7F

Ćwiczenie 5.1

Napisz prosty program, który ładuje wartość -9 do rejestru r16. Sprawdź reprezentację tej wartości za pomocą debugger'a. Spróbuj zapisać inne ujemne wartości. W tabeli poniżej podaj reprezentację binarną dla liczb -57 oraz -101.

Decimal	Binary	Hexadecimal
-57		
-101		

3.1 Wykrywanie przepełnienia w instrukcjach ze znakiem

W celu detekcji przepełnienia dla operacji arytmetycznych w kodzie naturalnym należy testować flagę 'carry' C procesor AVR.

W przypadku wystąpienia przepełnienia dla operacji arytmetycznych w U2 procesor AVR ustawia flagę V w rejestrze statusowym. Należy mieć na uwadze, iż procesor nie rozróżnia formatu zapisu argumentów wykonywanej operacji (czy zmienna ze znakiem czy bez znaku). Obowiązkiem programisty jest odpowiednia obsługa przepełnienia dla używanego formatu danych – flaga C dla liczb naturalnych, a flaga V dla liczb całkowitych.

Flaga V jest ustawiana wtedy kiedy spełniony jest jeden z dwóch poniższych warunków:

1. Występuje przeniesienie z bitu b6 na b7 (C7), oraz brak przeniesienia wyjściowego z bitu b7 (flaga C = 0)

2. Występuje przeniesienie wyjściowe z bitu b7 (flaga C=1) oraz brak przeniesienia z b6 do b7

(C7).

Czyli flaga przepełnienia (V) jest obliczana w następujący sposób: V = C7 ⊕ C.

Dla operacji na liczbach ze znakiem, flaga przepełnienia (V) oraz flaga N mogą być użyte do sprawdzane znaku wyniku. Flaga V=1 dla liczb ze znakiem oznacza, iż otrzymany wynik nie jest prawidłowy (wynik jest większy niż 127 lub mniejszy niż -128). Możliwe jest jednak ustalenie znaku wyniku za pomocą flagi S (Sign, S= N ⊕ V). Flaga S jest identyczna co do wartości z flagą N w przypadku kiedy flaga V jest wyzerowana. Sprawdzenie flagi S jest bezpieczniejszym sposobem na sprawdzenie znaku wyniku operacji, gdyż stan flagi S jest odporny na wystąpienie przepełnienia (flaga N nie jest).

W przypadku danych bez znaku flaga N oraz V może być ignorowana. Rezultat w przypadku operacji na danych bez znaku jest niepoprawny wtedy kiedy flaga C jest ustawiona (ustawienie flagi informuje, że wynik jest większy niż 255 i aby być prawidłowy musi być przechowywany na więcej niż 8 bitach). Ponadto flaga C jest użyteczna w przypadku realizacji operacji arytmetycznych na danych kilku bajtowych (informuje o możliwych przeniesieniach między bajtami).

Ćwiczenie 5.2

1. Napisz program:

ldi r20, 0x60 ;0110 0000 (+96)

ldi r21, 0x46 ;0100 0110 (+70)

add r20, r21 ;(+96) +(+70) = 1010 0110

(unsigned)(+96) + (unsigned)(+70) = (unsigned)(166) ***OK***

(signed)(+96) + (signed)(+70) = (signed)(-90) INVALID !!!

2. Uruchom i przekrokuj algorytm. Obserwuj stan flag C, V, N i S w symulatorze.

3. Sprawdź wyniki operacji dla innych argumentów: {70, 96}, {-70, -96}, {-126, 30}, {126, -6}, {-2, -5}.

Uwaga:

W asemblerze można bezpośrednio podawać argumenty o ujemnych wartościach, np.: *ldi r20, -70*

4. Zapisz wyniki operacji w poniższej tabelce:

Argumenty	**C**	**V**	**N**	**S**	**Wynik**		
					binary	**signed**	**unsigned**
{+70, +96}					0b.....		
{-70, -96}							
{-126, 30}							
{126, -6}							
{-2, -5}							

5. Zapisz program jako 'signed.asm'

4. Instrukcje skoków warunkowych

Flagi V, N, C oraz S posiadają bezpośrednio z nimi związane instrukcje względnego skoku warunkowego. Schemat tych instrukcji jest następujący:

branch_if_flag_cleared k ;-64 ≤ k ≤ +63

lub

branch_if_flag_set k ;-64 ≤ k ≤ +63

Instrukcja typu "Branch if flag cleared" testuje wartość flagi i wykonuje skok do podanego adresu jeżeli jest wyzerowana. Natomiast instrukcja "Branch if flag set" wykonuje skok jeżeli flaga jest ustawiona.

Instrukcja wykonuje skok do podanego adresu względnego, który znajduje się w zakresie: PC - 63 ≤ adres ≤ PC + 64). Parametr k stanowi offset przesunięcia względem aktualnej wartości licznika programu (PC) i jest zapisywany w kodzie uzupełnień do dwóch. Poniższa tabela podsumowuje instrukcje skoków warunkowych na podstawie wartości flag V, N, C oraz S.

	C ***Carry***	**V** ***Overflow***	**N** ***Negative***	**S** ***Sign***
Branch if cleared	***brcc*** *Branch if Carry cleared*	***brvc*** *Branch if Overflow Cleared*	***brmi*** *Branch if minus*	***brge*** *Branch if greater*
Branch if set	***brcs*** *Branch if Carry set*	***brvs*** *Branch if Overflow Set*	***brpl*** *Branch if plus*	***brlt*** *Branch if lower*

Przykład:

```
add r3,r4 ; Add r4 to r3
brvs overfl ; Branch if overflow
...
overfl: nop ; Branch destination (do nothing)
```

5. Operacja odejmowania

W wielu procesorach operacja odejmowania jest realizowana na dwa sposoby: standardowe odejmowanie bez przeniesienia oraz odejmowanie z „pożyczką". W procesorze AVR mamy pięć instrukcji realizujących operacje odejmowanie: *sub, sbc, subi, sbci* oraz *sbiw*. Operacje *sbc* i *sbci* to instrukcje odejmowania z „pożyczką" (używają flagi C).

5.1 Odejmowanie bez przeniesienia (*subtract - sub*)

Instrukcja odejmuje wartość znajdującą się w rejestrze źródłowym (Rr) od wartości znajdującej się w rejestrze docelowym (Rd) i umieszcza wynik odejmowania w rejestrze Rd. Instrukcja ustawia flagi C, V, N, S, Z rejestru statusowego.

sub Rd, Rr 0 ≤ d ≤ 31, 0 ≤ r ≤ 31

Przykład:

```
sub r13, r12    ; Subtract r12 from r13
brne noteq      ; Branch if r12<>r13
```

5.2 Odejmowanie z przeniesieniem (subtract with carry - *sbc*)

Instrukcja odejmuje wartość znajdującą się w rejestrze źródłowym (Rr) od wartości znajdującej się w rejestrze docelowym (Rd) i umieszcza wynik odejmowania w rejestrze Rd. Zawartość flagi C jest uwzględniana jako wartość zero lub jeden, która jest dodatkowo odejmowana od wyniku. Instrukcja ustawia flagi C, V, N, S, Z rejestru statusowego.

```
sbc Rd,Rr       ;0 ≤ d ≤ 31;     0 ≤ r ≤ 31;      Rd<-Rd-Rr-C
```

Operacja to umożliwia realizację odejmowania na argumentach kilku bajtowych (podobnie jak instrukcja adc)

Przykład:

```
                ;The 16-bit subtraction
                ; Subtract r1:r0 from r3:r2
sub r2, r0      ; Subtract low byte
sbc r3, r1      ; Subtract with carry, high byte
```

Ćwiczenie 5.2

Do powyższego kodu dodaj inicjalizację rejestrów stałymi wartościami: (r21:r20)=-318, (r23:r22)=271. Uruchom program, przekrokuj go i zaobserwuj wyniki.

Result		
Decimal	Binary	Hexadecimal

5.3 Odejmowanie stałej (subtract immediate - subi)

Instrukcja odejmuje od wartości znajdującej się rejestrze wartość stałą i umieszcza wynik odejmowania w rejestrze. Instrukcja ustawia flagi C, V, N, S, Z rejestru statusowego. Instrukcja operuje na rejestrach R16-R31.

```
subi Rd, K      ;16 ≤ d ≤ 31, 0 ≤ K ≤ 255
```

Przykład:

```
    subi r22, $11   ; Subtract $11 from r22
    brne noteq      ; Branch if r22<>$11
    ...
noteq: nop          ; Branch destination (do nothing)
```

5.4 Odejmowanie stałej z przeniesieniem (subtract immediate with carry - sbci)

Instrukcja odejmuje od wartości znajdującej się rejestrze wartość stałą z uwzględnieniem flagi C i umieszcza wynik odejmowania w rejestrze. Instrukcja ustawia flagi C, V, N, S, Z rejestru statusowego. Instrukcja operuje na rejestrach R16-R31.

sbci Rd,K ;16 ≤ d ≤ 31, 0 ≤ K ≤ 255

Przykład:

```
; Subtract $4F23 from r17:r16
subi r16, $23  ; Subtract low byte
sbci r17, $4F  ; Subtract with carry high byte
```

Ćwiczenie 5.3

Napisz program który odejmuje dwie liczby 16-bitowe. Wyświetl wartość bezwzględną wyniku na wyświetlaczu oraz sygnalizuj znak wyniku za pomocą pojedynczej wybranej diody LED. Dodatkowo, zrealizuj wykrywanie przepełnienia za pomocą wybranej diody kontrolnej LED.

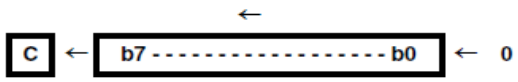
6. Instrukcje przesunięcia

Operacje przesunięcia w procesorze AVR operują na rejestrze ogólnego przeznaczenia i fladze przeniesienia C.

6.1 Instrukcja przesunięcia w lewo (logical shift left - lsl)

Operacja przesuwa wszystkie bity w podanym rejestrze o jeden bit w lewo. Najmłodszy bit jest zerowany. Najstarszy bit jest przesuwany do flagi C rejestru statusowego. Instrukcja ta wykonuje operacje mnożenia wartości ze znakiem (U2) oraz bez znaku przez dwa.

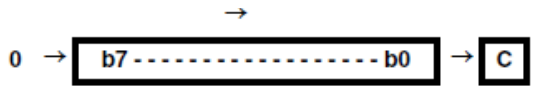
lsl Rd ;0 ≤ d ≤ 31



6.2 Instrukcja przesunięcia logicznego w prawo (logical shift right - lsr)

Operacja przesuwa wszystkie bity w podanym rejestrze o jeden bit w prawo. Najstarszy bit jest zerowany. Najmłodszy bit jest przesuwany do flagi C rejestru statusowego. Instrukcja ta wykonuje operacje dzielenia wartości bez znaku przez dwa.

lsr Rd ; 0 ≤ d ≤ 31



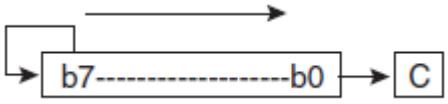
Przykład:

```
ldi r16, 0xA5          ;load hexadecimal 8-bit value from SRAM
mov r17, r16           ;r17=r16=0xA5
andi r17, 0x0f         ;isolate lower digit r17=0x05
out porta, r17         ;send lower hex digit to output port A (bits 0 to 3)
mov r17, r16           ;r17=0xA5 (0b10100101)
lsr r17                ;r17=0x52 (0b01010010)
lsr r17                ;r17=0x29 (0b00101001)
lsr r17                ;r17=0x14 (0b00010100)
lsr r17                ;r17=0x0A (0b00001010)
out porta, r17         ;send upper hex digit to output port A (bits 0 to 3)
```

6.3 Instrukcja przesunięcia arytmetycznego w prawo (arithmetic shift right - asr)

Instrukcja ta realizuje operacje arytmetycznego przesunięcia w prawo. Instrukcja może realizować dzielenie liczby ze znakiem (U2) przez dwa. Bity są przesuwane w prawo. Wyjątek stanowi bit znaku (najstarszy) który nie jest zmieniany. Najmłodszy bit przesuwany jest do flagi C.

Asr ; Rd 0 ≤ d ≤ 31



Przykład:

```
ldi r16,$10     ;Load decimal 16 into r16
asr r16         ;r16=r16 / 2
ldi r17,$fc     ;Load (-4) in r17
asr r17         ;r17=r17/2
```

Uwaga: Aby pomnożyć liczbę ze znakiem (U2) przez dwa można użyć instrukcji *lsl*. Flaga przepełnienia V oraz znaku S określają wartość (poprawność) wyniku.

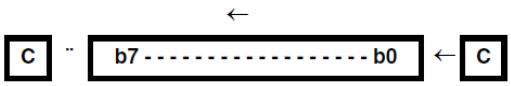
Przykład:

```
;Multiplication of signed numbers
                ;State after operation
ldi r20, -51    ;R20=11001101=(-51)         V=0 S=0 N=0 C=0
lsl r20         ;R20=10011010=(-102)        V=0 S=1 N=1 C=1
lsl r20         ;R20=00110100=52(wrong!)    V=1 S=1 N=0 C=0
stop: rjmp stop
```

6.4 Rotate Left trough Carry (rol)

Instrukcja przesuwa wszystkie bity w rejestrze Rd o jeden bit w lewo. Zawartość flagi C jest wstawiana do bitu 0 rejestru, a bit 7 jest wstawiany do flagi C.

rol Rd; 0 ≤ d ≤ 31



W połączeniu z operacją *lsl*, operacja *rol* pozwala na łatwą realizację mnożenia przez dwa wielbajtowej zmiennej ze znakiem (signed) lub bez znaku (unsigned), która jest umieszczona w kilku rejestrach.

Przykład:

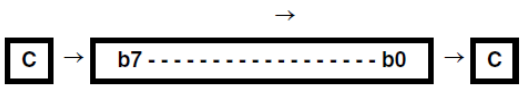
```
ldi r18, low(-1234) ; r19:r18 is a signed two-byte integer
ldi r19, high(-1234)
lsl r18 ; Multiply r19:r18 by two
rol r19 ; r19:r18 contains -2468
brvs overflow ; check V flag for overflow
```

```
                ….
                ldi r18, low(1234) ; r19:r18 is an unsigned two-byte integer
                ldi r19, high(1234)
                lsl r18 ; Multiply r19:r18 by two
                rol r19 ; r19:r18 contains 2468
                brcs overflow ; check C flag for overflow
                ….
overflow:       nop
```

6.5 Rotate Right trough Carry (ror)

Instrukcja przesuwa wszystkie bity w rejestrze Rd o jeden bit w prawo. Zawartość flagi C jest wstawiana do bitu 7 rejestru, a bit 0 jest wstawiany do flagi C.

ror Rd ; 0 ≤ d ≤ 31



W połączeniu z operacją a*sr*, operacja *ror* pozwala na łatwą realizację dzielenia przez dwa wielbajtowej zmiennej ze znakiem (signed), która jest umieszczona w kilku rejestrach. W połączeniu z operacją *lsr*, instrukcja dzieli przez dwa wielobajtową zmienną bez znaku (unsigned).

Example:

```
                ldi r18, low(1234) ; r19:r18 is an unsigned two-byte integer
                ldi r19, high(1234)
                lsr r19 ; Divide r19:r18 by two
                ror r18 ; r19:r18 is 617  (an unsigned two-byte integer)
                …
                ldi r18, low(-1234) ; r19:r18 is a signed two-byte integer
                ldi r19, high(-1234)
                asr r17 ; Divide r17:r16 by two
                ror r16 ; r17:r16 is -617 (a signed two-byte integer)
```

Ćwiczenie 5.4

Napisz program który zmienia wartość temperatury w stopniach Fahrenheit'a na stopnie Celsjusza.

1. Algorytm konwersji: **°C** = 5/9 (**°F** – 32).

2. Użyj 16-bitowej reprezentacji liczb ze znakiem.

3. Użyj operacji przesunięcia aby zrealizować mnożenie i dzielenie.

4. Wykrywaj przepełnienia.

5. Zapisz wynik operacji za pomocą wyświetlaczy 7 segmentowych oraz znak wartości za pomocą diody.

6. Zapisz program jako “C2F.asm”

Wskazówka:

Użyj następującej aproksymacji: $5/9 \approx 0.5625 = 1/2 + 1/16$