Wokół języka C

Asembler i adresowanie

część 1

W poprzednim numerze zamieszczone były bardzo ważne informacje na temat obsługi urządzeń peryferyjnych za pomocą rejestrów w przestrzeni I/O pamięci RAM. W niniejszym artykule podane są ogólne informacje o innych bardzo ważnych zagadnieniach – o rozkazach asemblera oraz o sposobach adresowania. Teoretycznie zagadnienia te nie mają bezpośredniego związku z językiem C, ale ich znajomość okaże się bardzo pożyteczna. Szersza wiedza usuwa strach i obawy oraz pomaga zrozumieć ścisły związek licznych poleceń języka C z asemblerem i z kodem maszynowym procesorów AVR. A w przyszłości być może w celu optymalizacji swoich programów w C zechcesz stosować w nich wstawki asemblerowe...

Grupy rozkazów

Jak już wiesz, w procesorach AVR rozkazy umieszczone w komórkach pamięci FLASH mają długość 16 bitów. Można powiedzieć, że rozkazy w kodzie maszynowym to 16-bitowe liczby dwójkowe. W kartach katalogowych procesorów AVR (na końcu) znajdziesz listę wszystkich rozkazów-instrukcji, ale nie w postaci liczb (dwójkowych czy szesnastkowych), tylko tzw. mnemoników – skrótów używanych w asemblerze.

Rysunek 1 pokazuje początek listy 131 rozkazów procesora ATmega328PB (str. 417 karty katalogowej). Liczba rozkazów (instrukcji), zależnie od „wielkości" procesora AVR, wynosi od kilkudziesięciu do stu kilkudziesięciu. Rozkazy te można podzielić na kilka głównych grup: arytmetyczno-logiczne, skoki, przesyłania danych oraz operacje na pojedynczych bitach. Omówmy je z grubsza.

Arytmetyczno-logiczne. Realizowane są w ***rejestrach o adresach 0...31***. Wynik operacji automatycznie modyfikuje też pewne bity (flagi) w rejestrze stanu (rejestr ***SREG***). Jak pokazuje rysunek 1, rozkazy arytmetyczne są zaskakująco nieliczne: dodawanie (ADD, ADC) i odejmowanie (SUB, SUBI, SBC, SBCI), a tylko w „najmocniejszych" procesorach proste mnożenie (MUL..., FMUL...). Więcej jest operacji logicznych na bitach całych rejestrów (AND, ANDI, OR, ORI, EOR, COM, NEG, SBR, CBR, INC, DEC, TST, CLR, SER).

Instruction Set Summary

ARITHMETIC AND LOGIC INSTRUCTIONS			
Mnemonics	Operands	Description	Operation
ADD	Rd, Rr	Add two Registers without Carry	Rd ← Rd + Rr
ADC	Rd, Rr	Add two Registers with Carry	Rd ← Rd + Rr + C
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K
SBC	Rd, Rr	Subtract two Registers with Carry	Rd ← Rd - Rr - C
SBCI	Rd, K	Subtract Constant from Reg with Carry.	Rd ← Rd - K - C
AND	Rd, Rr	Logical AND Registers	Rd ← Rd · Rr
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd · K
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr
COM	Rd	One's Complement	Rd ← 0xFF - Rd
NEG	Rd	Two's Complement	Rd ← 0x00 - Rd
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K
CBR	Rd,K	Clear Bit(s) in Register	Rd←Rd·(0xFF-K)
INC	Rd	Increment	Rd ← Rd + 1
DEC	Rd	Decrement	Rd ← Rd - 1
TST	Rd	Test for Zero or Minus	Rd ← Rd · Rd
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd
SER	Rd	Set Register	Rd ← 0xFF

Rys. 1

Skoki. W skokach chodzi o licznik programu (PC) i zaadresowane przezeń komórki ***pamięci FLASH***. „Podstawowy" wydaje się zawarty w dwóch komórkach skok (JMP, CALL), gdzie adres komórki FLASH, do której „program ma skoczyć", określony jest w drugiej komórce takiego „podwójnego" rozkazu. W mniejszych procesorach AVR wykorzystywane są „pojedyncze" rozkazy skoków (RJMP, IJMP, RCALL, ICALL). Oprócz tego istnieje szereg rozkazów ominięcia (*skip*) i skoków (*branch*) w zależności od różnych warunków (SBRC, SBRS, SBIC, SBIS i liczne BR...). Opis niektórych rozkazów związanych ze skokami znajdziesz na **rysunku 2**.

Przesyłanie danych. W tej grupie jedną ze „stron" przesyłania danych jest jeden z rejestrów roboczych R0...R31. W rozkazie MOV „drugą stroną" też jest jeden z tych rejestrów. Zdecydowana większość rozkazów (odmiany LD, LDS, ST, STS) dotyczy przesyłania bajtu danych w jednym lub drugim kierunku między rejestrem (R0...R31) a inną, dowolną komórką pamięci danych, zaadresowaną w jeden z rozmaitych sposobów. Na **rysunku 3** sposoby te przedstawione są w nieco dziwny sposób w ostatniej kolumnie. Istnieją „proste" rozkazy IN, OUT, służące do wymiany zawartości dowolnego z rejestrów R0...R31 oraz 64 „podstawowych" rejestrów przestrzeni I/O. Rozkaz LDI ładuje do jednego z rejestrów roboczych (R16...R31) ośmiobitową stałą, zawartą w kodzie rozkazu (czyli z pamięci FLASH do rejestru). Trzeba też wiedzieć, że istnieje rozkaz LPM (ELPM), który służy do przeniesienia danych trwale zapisanych w pamięci FLASH do rejestrów pamięci RAM.

Rys. 3

RJMP	al2	Skok względny	PC<-PC+al2+I
IJMP		Skok względny określony rejestrem Z	PC<-Z
RCALL	al2	Względne wywołanie podprogramu	PC<-PC+al2+I; (SP)<-PC+1
ICALL		Pośrednie wywołanie podprogramu	PC<-Z; (SP)<-PC+1
RET		Powrót z podprogramu	PC<-(SP)
RETI		Powrót z przerwania	PC<-(SP)
CPSE	Rd, Rs	Porównaj i skok, jeśli równe	(Rd=Rs)-> PC<-PC+(2/3)
CP	Rd, Rs	Porównaj rejestry	Rd-Rs
CPC	Rd, Rs	Porównaj rejestry wraz z C	Rd-Rs-C
CPI	Rh, K8	Porównaj rejestr ze stałą	Rh-K8
SBRC	Rs, b	Pomiń, gdy bit w rejestrze wyzerowany	(Rs.b=0)→ PC<-PC+I↳ PC<-PC+2
SBRS	Rs, b	Pomiń, gdy bit w rejestrze ustawiony	(Rs.b=I)→ PC<-PC+1↳ PC<-PC+2
SBIC	PI, b	Pomiń, gdy bit w rejestrze IO = 0	(PI.b=0)→ PC<-PC+I↳ PC<-PC+2
SBIS	PI, b	Pomiń, gdy bit w rejestrze IO ustawiony	(PI.b=I) → PC<-PC+1↳ PC<-PC+2
BRBS	b,k7	Skok, gdy flaga w SREG ustawiona	(SREG.b=I)→PC<-PC+k7+I ↳PC<-PC+I
BRBC	b,k7	Skok, gdy flaga w SREG skasowana	(SREG.b=0)→PC <-PC+k7+1 ↳PC <-PC+1
BREQ	k7	Skok względny, gdy równe	(Z=I)→PC<-PC+k7+I↳PC<-PC+I

Rys. 2

Przesyłanie danych

MOV	Rd, Rs	Kopiuj zawartość Rs do Rd	Rd<-Rs
LDI	Rh, K8	Ładuj rejestr stałą bezpośrednią	Rh<-K8
LDS	Rd, Al 6	Ładuj rejestr bezpośr. daną z SRAM	Rd<- (A16)
LD	Rd, X	Ładuj rejestr pośrednio daną z SRAM	Rd<- (X)
LD	Rd, X+	Ładuj rejestr pośrednio daną z SRAM	Rd<-(X); X<-X+I
LD	Rd,-X	Ładuj rejestr pośrednio daną z SRAM	X<-X-I;Rd<-(X)
LD	Rd, Y	Ładuj rejestr pośrednio daną z SRAM	Rd<- (Y)
LD	Rd, Y+	Ładuj rejestr pośrednio daną z SRAM	Rd<-(Y); Y<-Y+I
LD	Rd,-Y	Ładuj rejestr pośrednio daną z SRAM	Y<- Y-I;Rd<-(Y)
LDD	Rd, Y+K6	Ładuj rejestr pośrednio daną z SRAM	Rd<-(Y+K6)
LD	Rd, Z	Ładuj rejestr pośrednio daną z SRAM	Rd<- (Z)
LD	Rd, Z+	Ładuj rejestr pośrednio daną z SRAM	Rd<-(Z); Z<-Z+I
LD	Rd, -Z	Ładuj rejestr pośrednio daną z SRAM	Z<-Z-I; Rd<-(Z)
LDD	Rd,Z+K6	Ładuj rejestr pośrednio daną z SRAM	Rd<-(Z+K6)
STS	A16, Rs	Zachowaj bezpośr. rejestr w SRAM	(A16)<- Rs
ST	X, Rs	Zachowaj pośrednio rejestr w SRAM	(X)<- Rs
ST	X+, Rs	Zachowaj pośrednio rejestr w SRAM	(X)<- Rs; X<-X+I
ST	-X, Rs	Zachowaj pośrednio rejestr w SRAM	X<-X-I;(X)<-Rs
ST	Y, Rs	Zachowaj pośrednio rejestr w SRAM	(Y)<- Rs

Operacje bitowe. Jak pokazuje **rysunek 4**, istnieje kilka rozkazów przesuwania (*shift*) i zamiany (*swap*) bitów w rejestrach R0...R31 (co jest wykorzystywane m.in. do mnożenia/dzielenia przez potęgi liczby 2), a także szereg rozkazów ustawiania i zerowania pojedynczych bitów rejestru stanu SREG.

LSL	**Rd**	Przesuń logicznie w lewo	Rd(n+1) = Rd(n) Rd(0) = 0 C = Rd(7)
LSR	**Rd**	Przesuń logicznie w prawo	Rd(n) = Rd(n+1) Rd(7) = 0 C = Rd(0)
ROL	**Rd**	Rotuj w lewo przez Carry	Rd(n+1) = Rd(n) Rd(0) = C C = Rd(7)
ROR	**Rd**	Rotuj w prawo przez Carry	Rd(n) = Rd(n+1) Rd(7) = C C = Rd(0)
ASR	**Rd**	Przesuń arytmetycznie w prawo	Rd(7) = Rd(7) Rd(n) = Rd(n+1) C = Rd(0)
SWAP	**Rd**	Zamiana czwórek bitów	Rd(3-0) = Rd(7-4) Rd(7-4) = Rd(3-0)

Rys. 4

Jak pokazuje **rysunek 5**, wiele rozkazów dotyczy obsługi pojedynczych bitów w rejestrze stanu SREG. Istnieją też rozkazy SBI, CBI, które pozwalają bezpośrednio ustawiać i zerować pojedyncze bity w pierwszych 32 rejestrach I/O, co wykorzystywane jest m.in. do szybkiej obsługi pojedynczych pinów portów procesora. Natomiast do takich operacji na 32 „wyższych" rejestrach standardowej przestrzeni I/O oraz przestrzeni rozszerzonej (Extended I/O) trzeba stosować kombinację innych poleceń.

Bardzo pobieżnie zapoznaliśmy się z dostępnymi rozkazami. Jeśli chcesz, możesz znaleźć dodatkowe informacje o poleceniach kodu maszynowego i asemblera także na polskojęzycznych stronach, na przykład krótkie opisy:
http://goo.gl/18vaY8
http://goo.gl/X2Tkdh

Nie musisz wszystkiego rozumieć i wnikać w szczegóły. Gdy będziesz poznawać język C, zauważysz, że niektóre jego rozkazy są bezpośrednimi odpowiednikami rozkazów maszynowych. Choć C zaliczany jest do języków wysokiego poziomu, to jest on ściśle związany z kodem maszynowym, a to pozwala pisać w C zwięzłe i „szybkie" programy.

Rys. 5

SBI P, b	Ustaw bit w rejestrze I/O	I/O(P, b) := 1
CBI P, b	Skasuj bit w rejestrze I/O	I/O(P, b) := 0
BST Rr, b	Przepisz bit z rejestru do flagi T	T := Rr(b)
BLD Rd, b	Załaduj bit z flagi T do rejestru	Rd(b) := T
SEC	Ustaw flagę przeniesienia	C := 1
CLC	Zeruj flagę przeniesienia	C := 0
SEN	Ustaw flagę wartości ujemnej	N := 1
CLN	Skasuj flagę wartości ujemnej	N := 0
SEZ	Ustaw flagę zera	Z := 1
CLZ	Skasuj flagę zera	Z := 0
SEI	Włącz przerwania	I := 1
CLI	Wyłącz przerwania	I := 0
SES	Ustaw flagę znaku liczby	S := 1
CLS	Ustaw flagę znaku liczby	S := 0
SEV	Ustaw flagę przepełnienia	V := 1
CLV	Skasuj flagę przepełnienia	V := 0
SET	Ustaw flagę T	T := 1
CLT	Skasuj flagę T	T := 0
SEH	Ustaw flagę wewn. przeniesienia	H := 1
CLH	Skasuj flagę wewn. przeniesienia	H := 0
BSET s	Ustaw flagę	SREG(s) := 1
BCLR s	Skasuj flagę	SREG(s) := 0

Liczba rozkazów

Rozkazy (instrukcje) w kodzie maszynowym AVR to 16-bitowe liczby dwójkowe, więc sumaryczna liczba rozkazów wynosi 2^{16} =65536. W zapisie szesnastkowym są to czterocyfrowe liczby 0x0000...0xFFFF. Tymczasem na liście mamy poniżej 140 rozkazów AVR.

Na przykład rozkaz **NOP** (nic nie rób) ma numer 0000000000000000, czyli szesnastkowo 0x0000, rozkaz **RETI** (powrót z obsługi przerwania) ma numer 1001010100011000, czyli 0x9518, rozkaz **WDR** (wyzeruj licznik watchdoga) ma numer 1001010110101000, czyli 0x95A8.

Rys. 6

Czy wykorzystujemy poniżej 1% „możliwości"? Nie!

Wykorzystujemy ponad 97% rozkazów spośród 65536 możliwych. „Podstawowych" rozkazów mamy rzeczywiście tylko nieco ponad 130, ale w kodach większości rozkazów od razu zawarte są dodatkowe informacje (liczby, adresy). I tak na przykład rozkaz wymiany zawartości dwóch spośród rejestrów R0...R31 w asemblerze zapisujemy ogólnie jako: **MOV Rd, Rr**, podając w rozkazie dwa 5-bitowe numery: **Rr** rejestru źródłowego, **Rd** rejestru przeznaczenia (Rd ← Rr). Tak samo w rozkazie dodawania zawartości dwóch rejestrów: **ADD Rd, Rr** (Rd ← Rd + Rr). Analogicznie jest w innych rozkazach, gdzie w treści rozkazu występują inne liczby. Na przykład rozkaz **CPI** porównuje zawartość rejestru R16...R31 z liczbą 8-bitową (K), a wynik porównania trafia do rejestru stanu SREG.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	NOP
0	0	0	0	0	0	0	1	D	D	D	D	R	R	R	R	MOVW Rd,Rr Move register pair
0	0	0	0	0	0	1	0	d	d	d	d	r	r	r	r	MULS Rd,Rr
0	0	0	0	0	0	1	1	0	d	d	d	0	r	r	r	MULSU Rd,Rr
0	0	0	0	0	0	1	1	0	d	d	d	1	r	r	r	FMUL Rd,Rr
0	0	0	0	0	0	1	1	1	d	d	d	u	r	r	r	FMULS(U) Rd,Rr
0	0	0	c̄y̲	0	1	r		d	d	d	d	r	r	r	r	CPC/CP Rd,Rr
0	0	0	c̄y̲	1	0	r		d	d	d	d	r	r	r	r	SBC/SUB Rd,Rr
0	0	0	cy	1	1	r		d	d	d	d	r	r	r	r	ADD/ADC Rd,Rr ROL/LSL Rd (ADC/ADD with Rd=Rr)
0	0	0	1	0	0	r		d	d	d	d	r	r	r	r	CPSE Rd,Rr
0	0	1	0	0	0	r		d	d	d	d	r	r	r	r	AND Rd,Rr
0	0	1	0	0	1	r		d	d	d	d	r	r	r	r	EOR Rd,Rr
0	0	1	0	1	0	r		d	d	d	d	r	r	r	r	OR Rd,Rr
0	0	1	0	1	1	r		d	d	d	d	r	r	r	r	MOV Rd,Rr
0	0	1	1	K	K	K	K	d	d	d	d	K	K	K	K	CPI Rd,K
0	1	0	c̄y̲	K	K	K	K	d	d	d	d	K	K	K	K	SBCI/SUBI Rd,K
0	1	1	0	K	K	K	K	d	d	d	d	K	K	K	K	ORI Rd,K SBR Rd,K
0	1	1	1	K	K	K	K	d	d	d	d	K	K	K	K	ANDI Rd,K CBR Rd,K
1	0	k	0	k	k	s		d	d	d	d	y	k	k	k	LDD/STD through Z+k or Y+k
1	0	0	1	0	0	s		d	d	d	d	0	0	0	0	LDS rd,i/STS i,rd
16-Bit immediate SRAM-Address i																
1	0	0	1	0	0	s		d	d	d	d	y	0	0	1	LD/ST Rd through Z+/Y+
1	0	0	1	0	0	s		d	d	d	d	y	0	1	0	LD/ST Rd through -Z/-Y
1	0	0	1	0	0	0		d	d	d	d	0	1	q	0	LPM/ELPM Rd,Z
1	0	0	1	0	0	0		d	d	d	d	0	1	q	1	LPM/ELPM Rd,Z+
1	0	0	1	0	0	1		d	d	d	d	0	1	0	0	XCH Z,Rd
1	0	0	1	0	0	1		d	d	d	d	0	1	0	1	LAS Z,Rd
1	0	0	1	0	0	1		d	d	d	d	0	1	1	0	LAC Z,Rd
1	0	0	1	0	0	1		d	d	d	d	0	1	1	1	LAT Z,Rd
1	0	0	1	0	0	s		d	d	d	d	1	1	0	0	LD/ST Rd through X
1	0	0	1	0	0	s		d	d	d	d	1	1	0	1	LD/ST Rd through X+
1	0	0	1	0	0	s		d	d	d	d	1	1	1	0	LD/ST Rd through -X
1	0	0	1	0	0	s		d	d	d	d	1	1	1	1	POP/PUSH Rd
																1-operand instructions:
												0	0	0		COM
												0	0	1		NEG
1	0	0	1	0	1	0		d	d	d	d	0	1	0		SWAP
												0	1	1		INC
												1	0	1		ASR
												1	1	0		LSR
												1	1	1		ROR
1	0	0	1	0	1	0	0	B̄	b	b	b	1	0	0	0	SEx/CLx Status register clear/set bit
																Misc instructions:
								0	0	0	0					RET
								0	0	0	1					RETI
								1	0	0	0					SLEEP
1	0	0	1	0	1	0	1	1	0	0	1	1	0	0	0	BREAK

Rozkaz **SBI** ustawia jeden z siedmiu bitów (b) w jednym z 32 „pierwszych” rejestrów I/O o numerze A. W 16-bitowym kodzie maszynowym rozkazy

```
MOV Rd, Rr
ADD Rd, Rr
CPI Rd, K
SBI A, b
```

wyglądają następująco:

rozkaz	numer bitu w 16-bitowym rozkazie															
	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
	0...F				0...F				0...F				0...F			
MOV	0	0	1	1	1	1	r	d	d	d	d	d	r	r	r	r
ADD	0	0	0	0	1	1	r	d	d	d	d	d	r	r	r	r
CPI	0	0	1	1	K	K	K	K	d	d	d	d	K	K	K	K
SBI	1	0	0	1	1	0	1	0	A	A	A	A	A	b	b	b

Z uwagi na dodatkowe dane, w istocie mamy więc po 1024 rozkazy **MOV** i **ADD**, aż 4096 rozkazów **CPI** oraz 256 rozkazów **SBI**. Więcej informacji zawiera **rysunek 6**, będący fragmentem pełnej listy, którą można znaleźć w Elportalu oraz w Wikipedii na stronie: https://en.wikipedia.org/wiki/Atmel_AVR_instruction_set w skrócie: https://goo.gl/7dTF9i.

Warto zajrzeć na tę stronę i poznać dalsze szczegóły.

MOV – Copy Register

Rys. 7

This instruction makes a copy of one register into another. The source register Rr is left unchanged, while the destination register Rd is loaded with a copy of Rr.

Operation:

(i) Rd ← Rr

Syntax:	Operands:	Program Counter:
(i) MOV Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0010	11rd	dddd	rrrr

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        mov   r16,r0    ; Copy r0 to r16
        call  check     ; Call subroutine
        ...
check:  cpi   r16,$11   ; Compare r16 to $11
        ...
        ret             ; Return from subroutine
```

Words: 1 (2 bytes) **Cycles:** 1

lyons42.com/AVR/Opcodes/AVRAllOpcodes.html

Rys. 8

Summary - Blocks of 256 instructions

druga cyfra numeru rozkazu (hex) 0...F

pierwsza cyfra numeru rozkazu (hex)

	x0zz	x1zz	x2zz	x3zz	x4zz	x5zz	x6zz	x7zz	x8zz	x9zz	xAzz	xBzz	xCzz	xDzz	xEzz	xFzz
0xzz	255 [R]. nop	movw	muls	fmul, fmuls, fmulsu, mulsu	cpc				sbc				add			
1xzz	cpse				cp				sub				adc			
2xzz	and				eor				or				mov			
3xzz	cpi															
4xzz	sbci															
5xzz	subi															
6xzz	ori															
7xzz	andi															
8xzz	32 ld, 224 ldd		32 st, 224 std		ldd		std		ldd		std		ldd		std	
9xzz	(MISC)		112 [R]. 16 push, 112 st. 16 sts		(MISC)	(MISC)	adiw	sbiw	cbi	sbic	sbi	sbis	mul			
Axzz	ldd		std		ldd		std		ldd		std		ldd		std	
Bxzz	in								out							
Cxzz	rjmp															
Dxzz	rcall															
Exzz	ldi															
Fxzz	Conditional Branches				Conditional Branches				[R]. bld		[R]. bst		[R]. sbrc		[R]. sbrs	

lyons42.com/AVR/Opcodes/AVRAllOpcodes.html

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
2Cx	r16	r17	r18	r19	r20	r21	r22	r23	r24	r25	r26	r27	r28	r29	r30	r31
2FDx	mov r29, r16	mov r29, r17	mov r29, r18	mov r29, r19	mov r29, r20	mov r29, r21	mov r29, r22	mov r29, r23	mov r29, r24	mov r29, r25	mov r29, r26	mov r29, r27	mov r29, r28	mov r29, r29	mov r29, r30	mov r29, r31
2FEx	mov r30, r16	mov r30, r17	mov r30, r18	mov r30, r19	mov r30, r20	mov r30, r21	mov r30, r22	mov r30, r23	mov r30, r24	mov r30, r25	mov r30, r26	mov r30, r27	mov r30, r28	mov r30, r29	mov r30, r30	mov r30, r31
2FFx	mov r31, r16	mov r31, r17	mov r31, r18	mov r31, r19	mov r31, r20	mov r31, r21	mov r31, r22	mov r31, r23	mov r31, r24	mov r31, r25	mov r31, r26	mov r31, r27	mov r31, r28	mov r31, r29	mov r31, r30	mov r31, r31

mov - MOVe register to register

Rys. 9

Opcodes 30xx (0x3000 - 0x30FF)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
300x	cpi r16, 0x00	cpi r16, 0x01	cpi r16, 0x02	cpi r16, 0x03	cpi r16, 0x04	cpi r16, 0x05	cpi r16, 0x06	cpi r16, 0x07	cpi r16, 0x08	cpi r16, 0x09	cpi r16, 0x0A	cpi r16, 0x0B	cpi r16, 0x0C	cpi r16, 0x0D	cpi r16, 0x0E	cpi r16, 0x0F
301x	cpi r17, 0x00	cpi r17, 0x01	cpi r17, 0x02	cpi r17, 0x03	cpi r17, 0x04	cpi r17, 0x05	cpi r17, 0x06	cpi r17, 0x07	cpi r17, 0x08	cpi r17, 0x09	cpi r17, 0x0A	cpi r17, 0x0B	cpi r17, 0x0C	cpi r17, 0x0D	cpi r17, 0x0E	cpi r17, 0x0F
302x	cpi r18, 0x00	cpi r18, 0x01	cpi r18, 0x02	cpi r18, 0x03	cpi r18, 0x04	cpi r18, 0x05	cpi r18, 0x06	cpi r18, 0x07	cpi r18, 0x08	cpi r18, 0x09	cpi r18, 0x0A	cpi r18, 0x0B	cpi r18, 0x0C	cpi r18, 0x0D	cpi r18, 0x0E	cpi r18, 0x0F

Szczegółowy opis rozkazów rodziny AVR (po angielsku) jest na stronie: www.atmel.com/images/atmel-0856-avr-instruction-set-manual.pdf w skrócie: http://goo.gl/yA1RFY.

Rysunek 7 pokazuje opis rozkazu MOV, pochodzący z karty katalogowej. Pełne informacje o kodach wszystkich rozkazów (65536) można znaleźć na bardzo obszernej stronie: http://lyons42.com/AVR/Opcodes/AVRAllOpcodes.html w skrócie: http://goo.gl/8xZTEJ.

Stamtąd pochodzi też **rysunek 8**, pokazujący w zarysie, ile w rzeczywistości mamy poszczególnych rozkazów.

Z tejże strony pochodzi też **rysunek 9**, pokazujący niektóre rozkazy, których kody zaczynają się od 0x2F... (**MOV**) i 0x30... (**CPI**).

Jeśli chcesz, możesz poszukać w sieci dalszych informacji. Zachęcam, żebyś w ten sposób rozszerzył swoje horyzonty. Podkreślam jednak, że nie jest to niezbędne przy programowaniu w języku C. Niemniej chcąc programować mikroprocesory AVR, powinieneś przynajmniej z grubsza rozumieć tryby adresowania. Zajmiemy się tym za miesiąc.

Piotr Górecki