# Wokół języka C

część 2

# Asembler i adresowanie

Przed miesiącem omawialiśmy rozkazy asemblera oraz ich związek z zerami i jedynkami kodu maszynowego. Teraz omówimy **tryby adresowania**. Opis trybów adresowania i związane z tym rysunki straszą większość początkujących. Tymczasem zasady są proste. Zacznijmy od tego, że w ogóle ***adresowanie to podanie numeru komórki pamięci***, by odczytać jej zawartość albo coś do niej zapisać.

Jakiej pamięci?

Każdej dostępnej w procesorze: FLASH, RAM, EEPROM. Omówmy je po kolei.

## Adresowanie pamięci programu FLASH

W przypadku pamięci FLASH zasadniczo mamy do czynienia tylko z odczytem. O pracy procesora decyduje adresujący pamięć FLASH licznik programu (PC – *Program Counter*), który podczas normalnej pracy jest automatycznie inkrementowany (po wykonaniu rozkazu, zawartość PC jest zwiększana o 1). Dzięki temu standardowo wykonywane są rozkazy z kolejnych komórek pamięci FLASH. Ale jak już wiesz, do licznika programu można wpisać jakąś liczbę i w ten sposób wymusić przejście do innej części programu. Coś takiego automatycznie dzieje się w przypadku obsługi przerwań – przerwanie powoduje skok do komórki pamięci programu, przypisanej temu przerwaniu. Gdy przerwanie zostanie obsłużone, rozkaz RETI powoduje powrót do miejsca w programie tuż przed przerwaniem (ewentualnie wykorzystywane jest też stos oraz rozkazy PUSH i POP). Ale skoki mogą także następować z innych powodów.

W niektórych procesorach może być zrealizowany „podwójny” rozkaz skoku bezwarunkowego JMP albo przejścia do podprogramu CALL. Część adresu skoku zawarta jest w rozkazie, reszta – w następnej komórce pamięci. Struktura „podwójnego” rozkazu (długiego) skoku bezwarunkowego JMP jest następująca:

| 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 | | | |
| JMP | 1 | 0 | 0 | 1 | 0 | 1 | 0 | K | K | K | K | K | 1 | 1 | 0 | K |
| | K | K | K | K | K | K | K | K | K | K | K | K | K | K | K | K |

22-bitowa liczba **K** powala zaadresować 4194304 (4M) komórek pamięci programu. Taki bezpośredni sposób adresowania pamięci programu przedstawiony jest na **rysunku 10**. Rozkazy JMP, CALL nie są dostępne w wielu procesorach. Ale we wszystkich można realizować inne rozkazy skoków. Zawsze dostępne są „pojedyncze” rozkazy skoków względnych (*relative*) RJMP, RCALL:

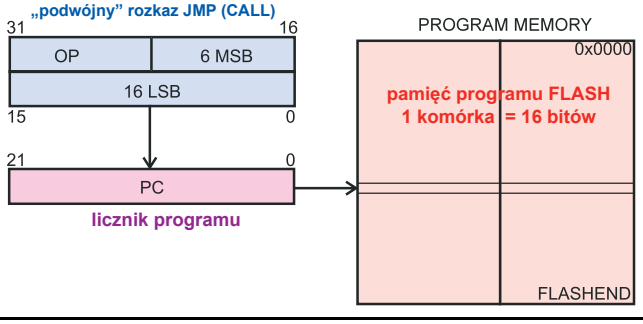
| rozkaz | numer bitu w 16-bitowym rozkazie | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
| | 0...F | | | | 0...F | | | | 0...F | | | | 0...F | | | |
| RJMP | 1 | 1 | 0 | 0 | S | S | S | S | S | S | S | S | S | S | S | S |

gdzie 12-bitowa liczba dwójkowa **S** (ze znakiem – w formacie uzupełnienia do 2) pozwala na skok o –2048...+2047 komórek pamięci względem aktualnego stanu licznika PC. Taki sposób adresowania pamięci jest zilustrowany na **rysunku 11**.

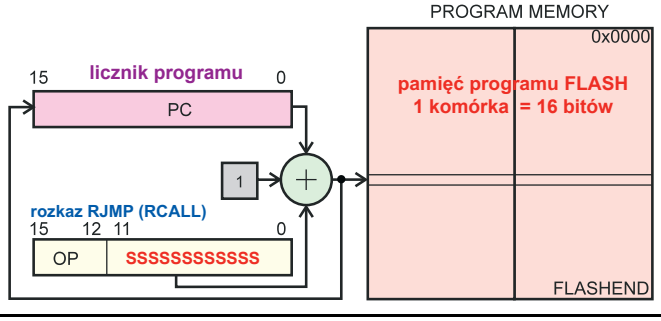
Dostępne są rozkazy IJMP, ICALL, gdzie wykorzystuje się tzw. *adresowanie pośrednie* – w sumie bardzo proste: do licznika programu PC wpisywana jest zawartość tzw. rejestru Z, a rejestr Z to nic innego jak para „ostatnich” rejestrów roboczych R31, R30, w sumie 16 bitów, co pozwala zaadresować 64K słów programu (w „większych” procesorach przewidziane są „rozszerzone” rozkazy EIJPM, EICALL, wykorzystujące też dodatkowy rejestr EIND).

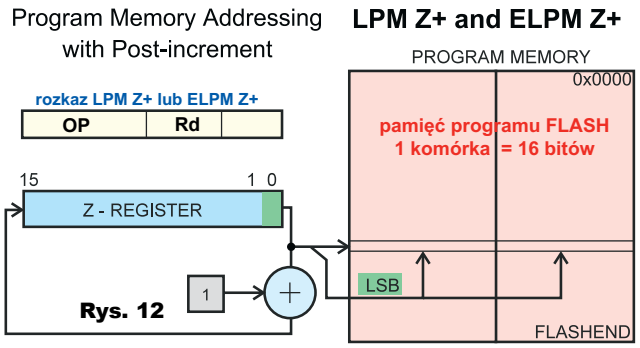
Omówione szczegóły dotyczą skoków, czyli sterowania przebiegiem programu. Programując w C, nie musisz się w to zbytnio zagłębiać. Powinieneś jednak wiedzieć o innej możliwości. Otóż pamięć RAM w procesorach AVR generalnie jest mała. Tymczasem w wielu urządzeniach wykorzystujemy wyświetlacze znakowe i graficzne. Wiele z wyświetlanej na nich treści to napisy lub inne znaki graficzne – w każdym razie są to niezmienne elementy (stałe). Innym przykładem są tablice przeliczeniowe (*look-up tables*). Standardowo te niezmienne elementy są „zaszyte” w programie w pamięci FLASH, ale po rozpoczęciu pracy programu tworzona jest ich kopia w pamięci RAM i dopiero stamtąd wysyłane są do wyświetlenia. Oznacza to marnotrawstwo skąpej pamięci RAM. Można temu zapobiec, wykorzystując dostępne w wielu procesorach trzy odmiany polecenia LPM (i „rozszerzone” ELPM z wykorzystaniem dodatkowego rejestru RAMPZ z przestrzeni I/O). Pozwalają one szybko i bezpośrednio odczytać zawartość komórek pamięci FLASH, gdzie nie są zawarte rozkazy zawierające „zaszyte stałe elementy”, tylko gdzie wspomniane stałe są zapisane „wprost”. I właśnie rozkazy LPM (ELPM) służą do odczytywania takich stałych danych zapisanych „wprost” z pamięci FLASH do jednego z rejestrów roboczych R0...R31. Problem jednak w tym, że komórki pamięci FLASH są 16-bitowe, a rejestr, do którego mają być odczytane, jest 8-bitowy. Jednorazowo można więc odczytać tylko jedną „połówkę” komórki

**Rys. 10** Direct Program Memory Addressing **JMP, CALL**

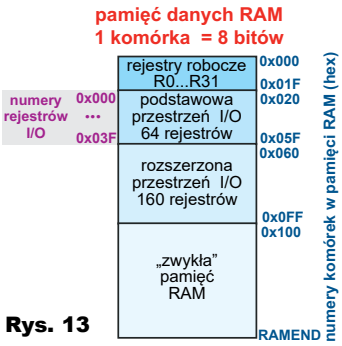



**Rys. 11** Relative Program Memory Addressing **RJMP, RCALL**






Rys. 12




Rys. 13

FLASH. Do określenia adresu i „połówki" odczytywanej komórki pamięci FLASH w rozkazach **LPM** i **LPM Rd,Z** wykorzystywana jest zawartość rejestru Z (+ ewentualnie RAMPZ). Najmłodszy bit w rejestrze Z określa, czy ma być odczytana „wyższa połówka", czy „niższa". Ponieważ zwykle nie chodzi o odczytanie jednego bajtu, tylko co najmniej kilku (np. liter do „napisów stałych" albo tablic przeliczeniowych look-up), dostępny jest też „seryjny" rozkaz **LPM Rd,Z+**, (**ELPM Rd,Z+**)

| 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 | | | |
| LMP RD,Z+ | 1 | 0 | 0 | 1 | 0 | 0 | 0 | d | d | d | d | d | 0 | 1 | 0 | 1 |
| ELMP RD,Z+ | 1 | 0 | 0 | 1 | 0 | 0 | 0 | d | d | d | d | d | 0 | 1 | 1 | 1 |

gdzie zawartość rejestru Z jest automatycznie zwiększana po każdym odczycie, jak pokazuje **rysunek 12**. Pięć bitów **d** określa numer rejestru, dokąd trafią odczytane dane. Oczywiście z uwagi na to, czym jest rejestr Z (R31+R30), mało sensowne są rozkazy **LPM r30,Z+** oraz **LPM r31,Z+**.

## Adresowanie pamięci danych RAM

Więcej rodzajów adresowania dotyczy pamięci RAM. Tym bardziej że pamięć RAM dzielimy na: 32 rejestry robocze (R0...R31), 64 rejestry podstawowej przestrzeni I/O, dodatkowo w „większych" procesorach 160 rejestrów rozszerzonej przestrzeni I/O, a dalej „zwykłą" pamięć RAM – **rysunek 13**. Komórki RAM mają numerację ciągłą, ale 64 rejestry podstawowej przestrzeni I/O mają oddzielną numerację – numer rejestru jest mniejszy o 32 (0x20) od numeru komórki RAM.

Podczas pracy procesora najczęściej wykorzystujemy rejestry robocze R0...R31, ponieważ są one ściśle związane z „kalkulatorem" CPU. Dlatego w operacjach przesyłania danych jedną ze stron jest właśnie któryś z 32 rejestrów roboczych. Aby go zaadresować, trzeba podać jego numer: wystarczy do tego 5 bitów ($2^5 = 32$). Numer wykorzystywanego rejestru roboczego zawarty jest w treści rozkazu.

Najprostszym sposobem adresowania jest więc umieszczenie w rozkazie adresu rejestru albo adresów dwóch rejestrów, jak na przykład w **MOV Rd, Rr**:

| MOV | 0 | 0 | 1 | 1 | 1 | 1 | r | d | d | d | d | d | r | r | r | r |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

– omawialiśmy to dość dokładnie w poprzednim miesiącu. Ale tak adresujemy tylko rejestry robocze w pierwszych 32 komórkach RAM, wykorzystując ich pięciobitowe adresy-numery.

Natomiast rozkazy **IN, OUT**:

| IN | 1 | 0 | 1 | 1 | 0 | a | a | d | d | d | d | d | a | a | a | a |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OUT | 1 | 0 | 1 | 1 | 1 | a | a | d | d | d | d | d | a | a | a | a |

wymieniają dane między rejestrem roboczym Rd (**d**) a jednym z 64 podstawowych rejestrów I/O o sześciobitowym numerze (**a**). Zauważ, że w kodzie rozkazów **IN, OUT** zawarty jest numer kolejny rejestru I/O, a nie jego adres RAM, który jest o 32 (0x20) większy – stąd też podwójna numeracja tych podstawowych rejestrów I/O w kartach katalogowych. Ilustruje to **rysunek 14**.

Dostępne są też pokrewne „podwójne" rozkazy **LDS** i **STS**, wykorzystujące adresowanie bezpośrednie. Pozwalają one odpowiednio wczytać do rejestru roboczego Rd (**d**) zawartość komórki dowolnej pamięci RAM o 16-bitowym adresie-numerze (**K**) oraz zawartość rejestru wpisać do tak zaadresowanej komórki pamięci:

| 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 | | | |
| LDS | 1 | 0 | 0 | 1 | 0 | 0 | 0 | d | d | d | d | d | 0 | 0 | 0 | 0 |
| | K | K | K | K | K | K | K | K | K | K | K | K | K | K | K | K |
| STS | 1 | 0 | 0 | 1 | 0 | 0 | 1 | d | d | d | d | d | 0 | 0 | 0 | 0 |
| | K | K | K | K | K | K | K | K | K | K | K | K | K | K | K | K |

Ilustruje to **rysunek 15**.

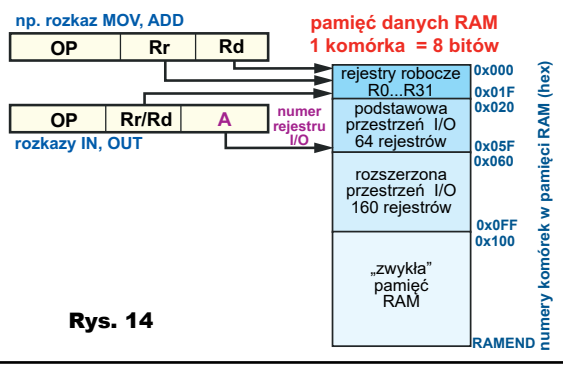
Ogólnie biorąc, przy adresowaniu bezpośrednim, numery-adresy komórek pamięci RAM podane są w treści rozkazu. I to jest w sumie proste. Początkującym sprawia kłopot zrozumienie, co to jest...

**Adresowanie pośrednie** (*indirect adressing*), choć jego idea też wcale nie jest trudna. Znów jedną ze „stron" wymiany danych jest któryś rejestr roboczy R0... R31 (jego pięciobitowy adres jest podany w treści rozkazu, czyli w pamięci FLASH). „Drugą stroną" operacji wymiany danych jest dowolna komórka pamięci RAM o numerze-adresie, który NIE jest podany w treści rozkazu, tylko zawarty jest w jednym z 16-bitowych rejestrów X, Y, Z. Rejestry wskaźnikowe X, Y, Z to w rzeczywistości pary 8-bitowych rejestrów roboczych, odpowiednio R26-R27; R28-R29; R30-R31. Czyli przy adresowaniu pośrednim ***adres potrzebnej komórki RAM podany jest w innej, określonej części pamięci RAM***. W katalogach przedstawione to jest jak na **rysunku 16a**.

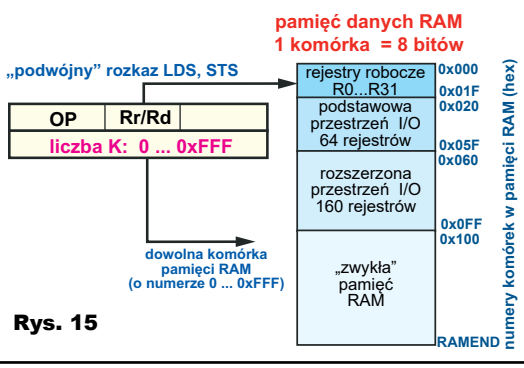
To, że adres potrzebnej komórki RAM jest zawarty w najwyższych rejestrach roboczych X, Y lub Z, pokazuje **rysunek 16b**, przedstawiający nieco dokładniej działanie „podstawowych" rozkazów odczytu **LD Rs,X**, **LD Rs,Y**, **LD Rs,Z** oraz zapisu dowolnej komórki pamięci **ST Rr,X**, **ST Rr,Y**, **ST Rr,Z**.

Jednak sposób prezentacji z rysunku 16b utrudnia pokazanie zasad działania odmian rozkazów LD, ST, na przykład **LD Rs,-X**, **LD Rs,X+** (czy też LDD i STD).

*Ciąg dalszy na stronie 65*




Rys. 14




Rys. 15