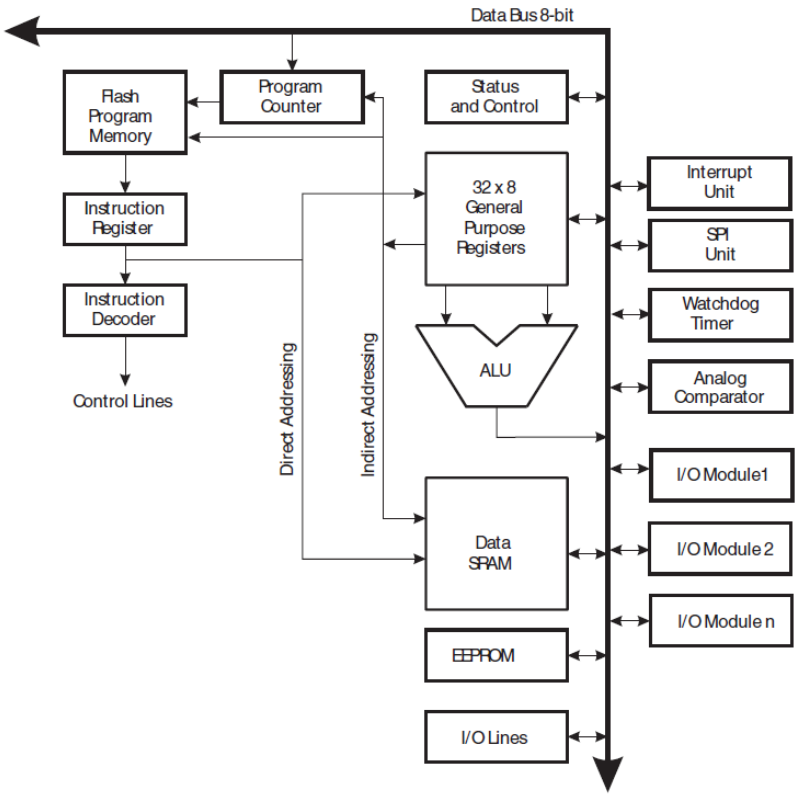
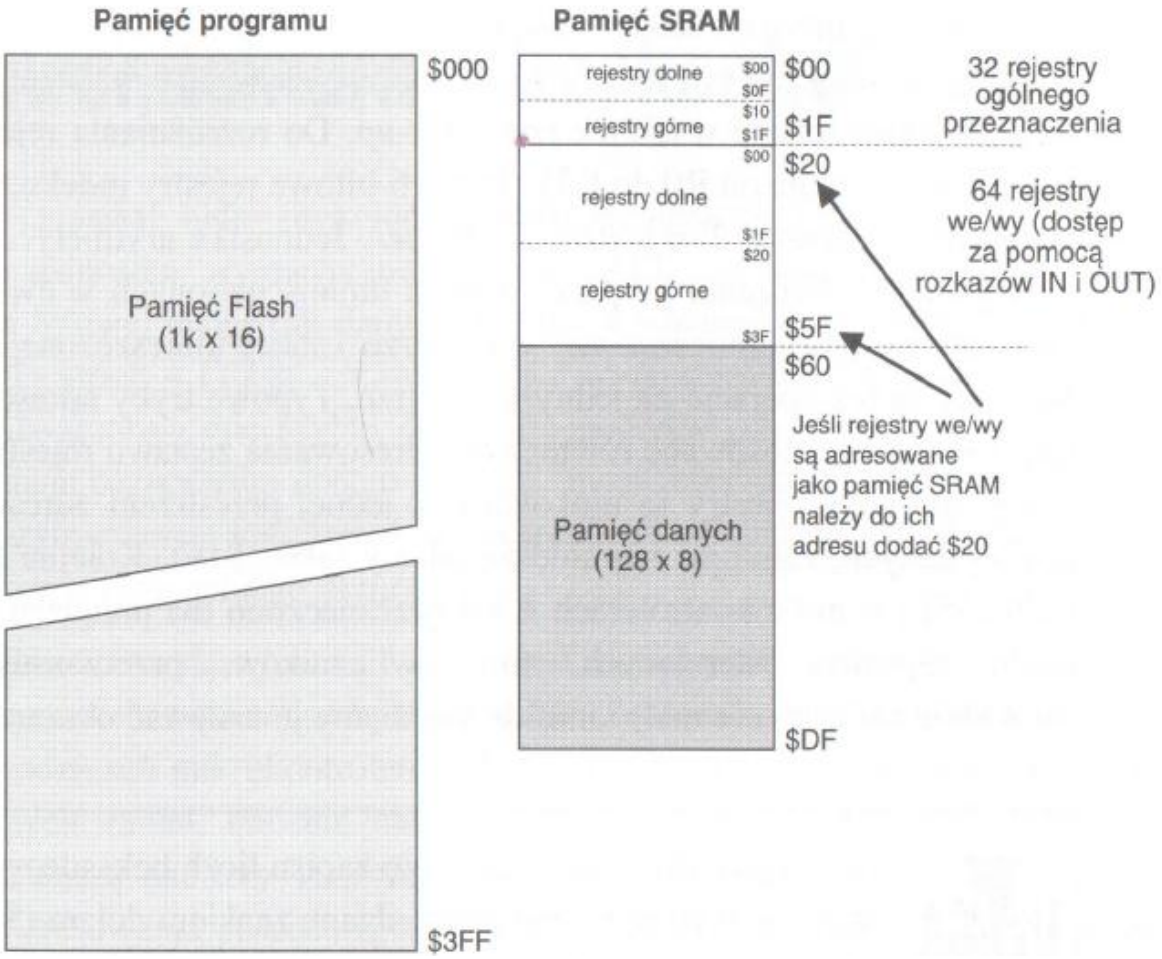
Schemat blokowy architektury mikrokontrolerów ATMega



Pamięć mikrokontrolera ATMega256

Procesor ten zrealizowany jest w architekturze harwardzkiej: posiada 256 kB pamięci Flash przeznaczonej dla instrukcji programu, w osobnej przestrzeni adresowej 8 kB pamięci SRAM przeznaczonej na przechowywanie danych oraz 4kB pamięci EEPROM traktowanej jako układ peryferyjny.

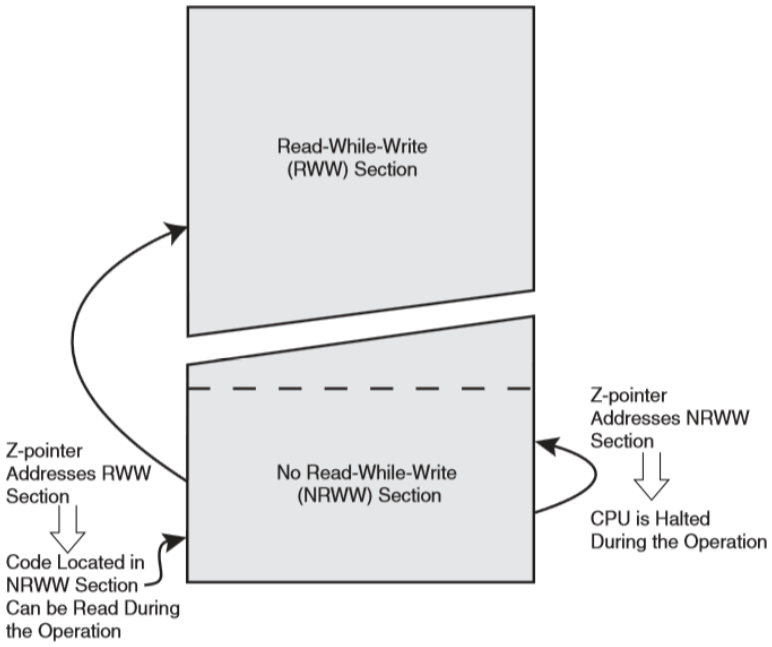


• Wewnętrzna reprogramowalna pamięć programu typu Flash.

ATmega256 zawiera 256kB reprogramowalnej pamięci Flash, którą można programować w systemie. Ponieważ wszystkie rozkazy są 16-to lub 32 bitowe, pamięć Flash jest zorganizowana jako 128K x 16. Dla bezpieczeństwa oprogramowania pamięć Flash jest podzielona na dwie sekcje: Inicjującą system (Boot Program) i dla programów użytkowych. Pamięć Flash ma trwałość ponad 10000 cykli zapisu i kasowania. Licznik programu w ATmega256 jest 17 bitowy, tak więc może zaadresować on 128kseów pamięci programu. Szczegóły dotyczące sekcji inicjującej system i związane z nim bity zabezpieczeń zostały opisane poniżej. W pamięci Flash mogą być również umieszczane tablice stałych dostępne za pomocą instrukcji LPM (Load Program Memory).

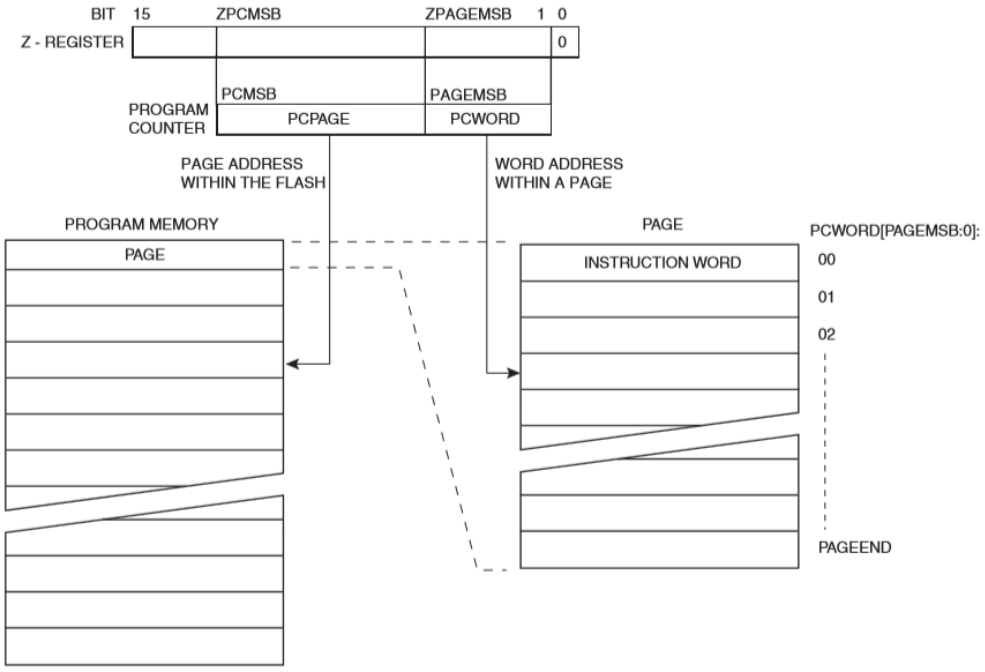
Uruchomienie programu bootloadera może zostać zainicjowane przez wykonanie skoku do tego obszaru lub wywołanie procedury. Można też skorzystać z sygnału Reset, którego obsługa może polegać na wywołaniu programu ładującego. Ustawia się w tym celu specjalne bity sterujące, których nie można zmienić później programowo lecz tylko programując kontroler przez interfejs programowania szeregowego lub równoległego.

Figure 29-1. Read-While-Write vs. No Read-While-Write



Programowanie pamięci Flash następuje za pomocą instrukcji **SPM** (Store Program Memory) lub **ESPM** (dla pamięci większej niż 64KB), stronami, z wykorzystaniem bufora strony. Wcześniej należy skasować zawartość strony pamięci. Do adresowania wykorzystuje się rejestr indeksowy Z, którego starsze bity określają numer strony w pamięci zaś młodsze bity – adres słowa na stronie. Wartość zapisywana w pamięci pobierana jest z pary rejestrów R1:R0. Podczas programowania należy kontrolować przerwania oraz sprawdzać czy nie trwa zapis do pamięci EEPROM, gdyż może to zakłócić operację zapisu do pamięci Flash.

Figure 29-3. Addressing the Flash During SPM[(1)]



Note: 1. The different variables used in Figure 29-3 are listed in Table 29-9 on page 329.

Bity Boot Lock Protection Modes pozwalają lub blokują możliwość programowania wybranych obszarów pamięci Flash za pomocą instrukcji SPM.

Odczyt stałych z pamięci Flash dokonywany jest za pomocą instrukcji **LPM** lub **ELPM** i rejestru RAMPZ (RAM Page Z Select Register) korzystających z adresowania pośredniego przez rejestr Z, np.

```
ldi ZH, high(Stala*2+1)     ;ładuj MSB wskaźnika Z starszą częścią adresu stałej
ldi ZL, low(Stala*2+1)      ;ładuj MSB wskaźnika Z młodszą częścią adresu stałej
lpm R0, Z                   ;ładuj rejestr R0 wartością 0x12
….
Stala: .dw 0x12AB           ;deklaracja dwubajtowej stałej w pamięci programu
```

Adresowanie skoków i wywołania podprogramów.

Dostępne są w tym przypadku trzy tryby adresowania: bezpośredni, pośredni i względny.

Z uwagi na prędkość działania najczęściej wykorzystuje się adresowanie względne. Tak działają **rjmp** – skok względny i **rcall** – wywołanie podprogramu. Rozkazy te, o długości

jednego słowa, posiadają argument dodawany do aktualnej wartości licznika programu co pzwala przenieść się w zakresie -2048…2047 słów.

Ograniczenia tego nie posiadają rozkazy **jmp** oraz **call**, które mają długość dwóch słów i zawierają adres bezpośredni słowa, do którego chcemy przenieść sterowanie.

Trzeci sposób adresowania wykorzystywany jest przez instrukcję **ijmp** oraz **icall**. Tryb ten, pośredniego adresowania pamięci programu, wykorzystuje rejestr indeksowy Z w roli wskaźnika. Pozwala to wybrać podprogram do którego przenoszone jest sterowanie.

```
        ldi   ZH, high(Wkonaj)
        ldi   ZL, low(Wykonaj)
        icall
        ….
Wykonaj:                    ; Początek podprogramu Wykonaj
        ….
        ret
```

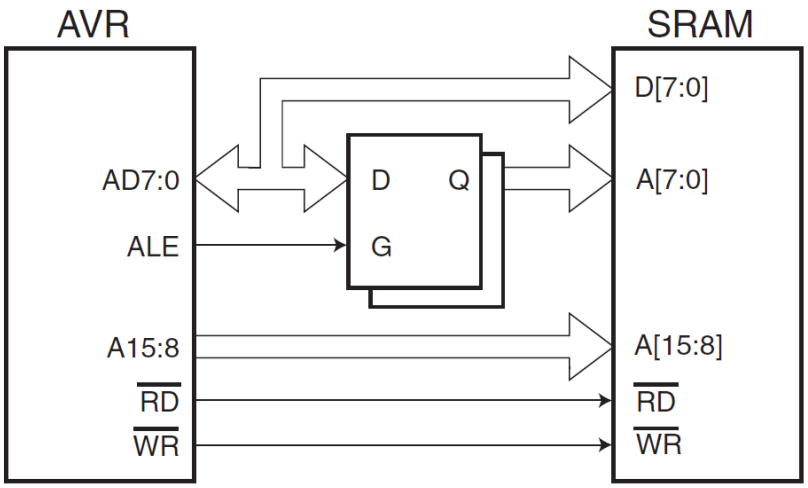
• pamięć danych typu SRAM.

Obszar pamięci SRAM połączono z rejestrami roboczymi i funkcyjnymi, tworząc wspólną przestrzeń adresową. Umożliwia to jednakowy sposób adresowania wszystkich trzech obszarów.

Data Memory Map

Address (HEX)	
0 - 1F	32 Registers
20 - 5F	64 I/O Registers
60 - 1FF	416 External I/O Registers
200 21FF	Internal SRAM (8192 × 8)
2200	External SRAM (0 - 64K × 8)
FFFF	

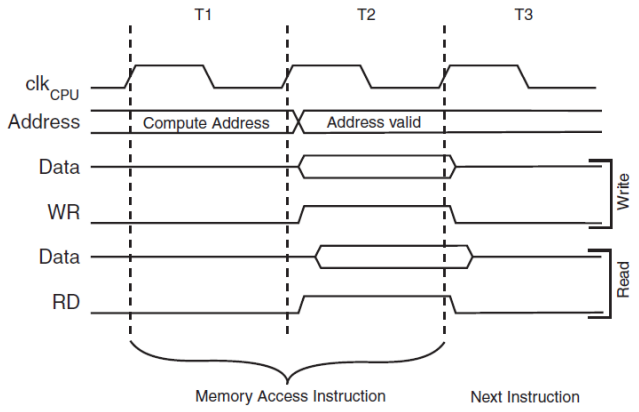
Figure 9-2. External SRAM Connected to the AVR



8.2.1 Data Memory Access Times

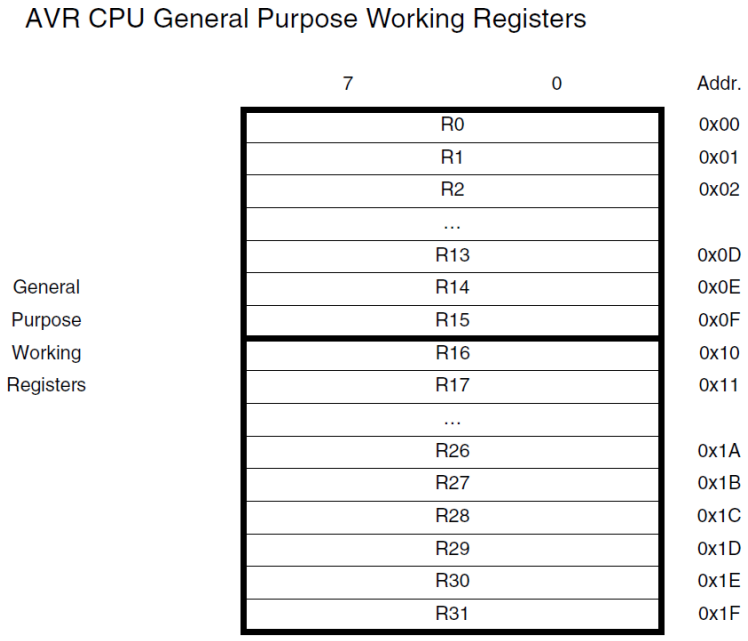
This section describes the general access timing concepts for internal memory access. The internal data SRAM access is performed in two clk_{CPU} cycles as described in Figure 8-3.

Figure 8-3. On-chip Data SRAM Access Cycles

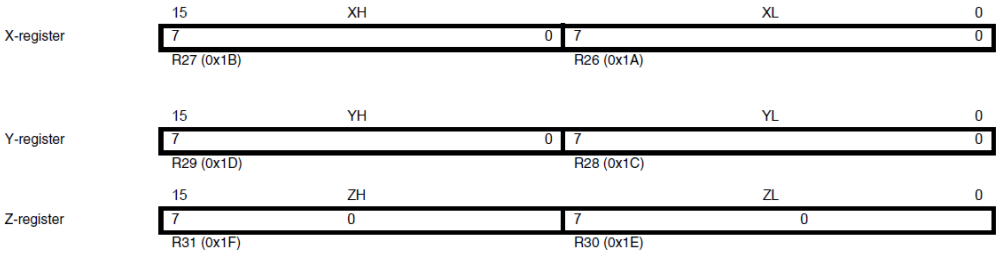


Procesor posiada 32-bajtowy obszar rejestrów roboczych, do których jednostka arytmetyczno-logiczna ma bezpośredni dostęp. Jest to wymuszone przez architekturę RISC i wymagania języków wysokiego poziomu (języka C). Dzięki możliwości wykorzystania każdego z tych rejestrów w funkcji akumulatora unika się wielu operacji przenoszenia danych, co pozwala przyspieszyć działanie i zmniejszyć objętość tworzonych programów.

Na rysunku poniżej przedstawiono organizację przestrzeni adresowej rejestrów roboczych i ich nazwy. Sześć ostatnich rejestrów (R26 do R31) tworzy trzy pary dwubajtowych rejestrów indeksowych używanych jako wskaźniki (X, Y, Z).



Rozmieszczenie rejestrów ogólnego przeznaczenia



Budowa logiczna rejestrów indeksowych X, Y, Z

Uwaga: Rejestry R0..R15 objęte zostały ograniczeniem, przez co nie mogą być one używane przez ALU przy operacjach ze stałymi ładowanymi bezpośrednio (ang. immediate). Rozkazami, które nie mogą być wiązane z tą częścią pamięci są: ldi (ładowanie stałej), subi, sbci, andi, ori i cpi.

Dostęp do pamięci SRAM (poza rejestrami) trwa dwa cykle zegara, jednak nie można na niej przeprowadzać operacji arytmetyczno-logicznych w sposób bezpośredni.

Pamięć SRAM poza miejscem przechowywania danych stanowi jednocześnie stos mikrokontrolera. Stos służy do pamiętania adresu powrotu z podprogramu, przekazywania argumentów oraz przechowywania zmiennych lokalnych.

Pomiędzy rejestrami roboczymi a pamięcią SRAM znajduje się przestrzeń wejścia-wyjścia w której znajdują się wszelkie rejestry funkcyjne, nazywane także specjalnymi. Dostarczają one informacji o aktualnym stanie mikrokontrolera lub umożliwiają sterowanie pracą rdzenia i wewnętrznych układów peryferyjnych.

Tryby adresowania danych.

Cały obszar pamięci danych może być adresowany bezpośrednio lub pośrednio. Sposób bezpośredni polega na umieszczeniu w rozkazie **lds** lub **sts** dwubajtowego adresu pod jakim w pamięci danych znajduje się zmienna przeznaczona do odczytu lub zapisu.

Adresowanie pośrednie wymaga w pierwszej kolejności załadowanie wybranego wskaźnika (X, Y lub Z) adresem komórki, w której znajduje zmienna. Dopiero wówczas możliwy jest dostęp do niej za pomocą instrukcji **ld** lub **st**.

```
ldi  XH, high(TAB)

ldi  XL, low(TAB)

st  X, R0
```

Dopisując po symbolu rejestru indeksowego znak + lub – przed nim, możemy otrzymać tzw. adresowanie pośrednie z postinkrementacją lub predekrementacją.

Przestrzeń wejścia-wyjścia.

Przestrzeń wejścia-wyjścia możemy adresować na te same sposoby, które wykorzystujemy do adresowania pamięci SRAM. Pamiętać należy jedynie, aby bity określane w dokumentacji danego rejestru funkcyjnego jako zarezerwowane wypełniać przy zapisie zerami. Takie rozwiązanie nie jest szczególnie wygodne i istnieje możliwość adresowania bezpośredniego poprzez jednotaktowe instrukcje **in** oraz **out**.

Uwaga: Przy użyciu rozkazów in i out odwołujemy się do przestrzeni wejścia-wyjścia tak, jakby obszar ten był osobnym zbiorem rejestrów zawartych w przedziale adresowym 0x00…0x3F. Jeśli chcemy dostać się do tej części pamięci jak do części pamięci danych do adresu wybranego rejestru należy dodać wartość 0x20 (przeskakujemy 32 rejestry robocze). W praktyce najczęściej odwołujemy się do rejestrów wejścia-wyjścia przez instrukcje in oraz out, posługując się w tym celu predefiniowanymi nazwami symbolicznymi zawartych tam rejestrów.

Przykład:

```
in  R0, SREG        lub        lds R0, SREG+0x20
```