Programowanie mikrokontrolerów

Asembler AVR, część 2

Marcin Engel Marcin Peczarski

Instytut Informatyki Uniwersytetu Warszawskiego

6 października 2012

Przesłania między rejestrami

- Między dowolnymi dwoma rejestrami

  `MOV  Rd, Rr`

- Między parami rejestrów

  `MOVW Rd, Rr`

- Przykład: przesłanie wyniku mnożenia do pary rejestrów `R17:R16`

  `MOVW R17:R16, R1:R0`

Adresowanie natychmiastowe

- Inicjowanie rejestru dowolną stałą, tylko rejestry `R16`, ..., `R31`, nie modyfikuje znaczników.

```
LDI  Rd, K
```

- Wyzerowanie dowolnego rejestru, modyfikuje znaczniki.

```
CLR  Rd          ; inny zapis dla EOR Rd, Rd
```

- Ustawienie wszystkich bitów na 1, tylko rejestry `R16`, ..., `R31`, nie modyfikuje znaczników.

```
SER  Rd          ; inny zapis dla LDI Rd, $FF
```

Adresowanie bezpośrednie i pośrednie

- Ładowanie rejestru wartością z pamięci danych

  `LDS  Rd, k`

- Zapisanie wartości z rejestru do pamięci danych

  `STS  k, Rr`

- Ładowanie rejestru wartością z pamięci danych, adres w rejestrze `X`, `Y` lub `Z`

  `LD   Rd, Rxyz`

- Zapisanie wartości z rejestru do pamięci danych, adres w rejestrze `X`, `Y` lub `Z`

  `ST   Rxyz, Rr`

Adresowanie pośrednie z postinkrementacją i predekrementacją

- Ładowanie rejestru wartością z pamięci danych, adres w rejestrze `X`, `Y` lub `Z`, który jest następnie zwiększany o 1.

  ```
  LD   Rd, Rxyz+
  ```

- Zapisanie wartości z rejestru do pamięci danych, adres w rejestrze `X`, `Y` lub `Z`, który jest następnie zwiększany o 1.

  ```
  ST   Rxyz+, Rr
  ```

- Ładowanie rejestru wartością z pamięci danych, adres w rejestrze `X`, `Y` lub `Z`, który jest uprzednio zmniejszany o 1.

  ```
  LD   Rd, -Rxyz
  ```

- Zapisanie wartości z rejestru do pamięci danych, adres w rejestrze `X`, `Y` lub `Z`, który jest uprzednio zmniejszany o 1.

  ```
  ST   -Rxyz, Rr
  ```

Adresowanie pośrednie z przemieszczeniem

- Ładowanie rejestru wartością z pamięci danych, adres w rejestrze `Y` lub `Z` plus przemieszczenie o wartości od 0 do 63.

  ```
  LDD  Rd, Ryz+q
  ```

- Zapisanie wartości z rejestru do pamięci danych, adres w rejestrze `Y` lub `Z` plus przemieszczenie o wartości od 0 do 63.

  ```
  STD   Ryz+q, Rr
  ```

Adresowanie pamięci programu

- Załadowanie do rejestru bajtu z pamięci programu o adresie w rejestrze `Z`

  ```
  LPM  Rd, Z
  ```

- Załadowanie do rejestru bajtu z pamięci programu o adresie w rejestrze `Z`, następnie zwiększenie `Z`

  ```
  LPM  Rd, Z+
  ```

- Zapisanie zawartości pary rejestrów `R1:R0` w pamięci programu o adresie w rejestrze `Z`

  ```
  SPM
  ```

- Rozkaz `LPM` adresuje bajty.
- Rozkaz `SPM` adresuje słowa.
- Rozkaz `SPM` umożliwia samomodyfikowanie się programu. Aby poznać dokładnie jego działanie, zajrzyj do dokumentacji.

Adresowanie pamięci programu

- Przykład: załadowanie do rejestru `Z` 16-bitowej wartości z pamięci programu

```
    LDI  ZL, LOW(VALUE << 1)
    LDI  ZH, HIGH(VALUE << 1)
    LPM  R16, Z+
    LPM  R17, Z
    MOVW ZH:ZL, R17:R16

.CSEG
VALUE:
    .DW  3456
```

Operacje stosowe

- Odłożenie na stos zawartości rejestru

```
PUSH Rr
```

- Zdjęcie wartości ze stosu

```
POP  Rd
```

- Przykład: odłożenie na stos rejestru stanu na początku procedury obsługi przerwania

```
PUSH R16
IN   R16, SREG
PUSH R16
```

- Przykład: odtworzenie rejestru stanu na końcu procedury obsługi przerwania

```
POP  R16
OUT  SREG, R16
POP  R16
```

Translacja warunkowa

- Sterowanie translacją umożliwiają dyrektywy:

```
.IF    .ELIF    .ELSE    .ENDIF
.IFDEF    .IFNDEF
```

- Przykład: chcemy uzależnić kod od środowiska, w którym kompilujemy nasz program.

```
.EQU VMLAB = 0 ; definiujemy 0 lub 1

.CSEG
.IF VMLAB == 1
        RJMP    MIDDLE_VALUE
.ELSE
        RJMP    TOP_VALUE
.ENDIF
```

Makra

- Przykład: odczyt bajtu z EEPROM

```
.MACRO  LDE
        LDI     @0, HIGH(@1)
        OUT     EEARH, @0
        LDI     @0, LOW(@1)
        OUT     EEARL, @0
        LDI     @0, 1 << EERE
        OUT     EECR, @0
        IN      @0, EEDR
.ENDMACRO
```

- Przykład: użycie tego makra

```
.ESEG
        CV1:    .DB 4
.CSEG
        LDE     R17, CV1
```

Mnożenie

- Mamy następujące rozkazy mnożenia:

```
MUL    Rd, Rr
MULS   Rd, Rr
MULSU  Rd, Rr
FMUL   Rd, Rr
FMULS  Rd, Rr
FMULSU Rd, Rr
```

- Każdy z rozkazów mnoży dwie 8-bitowe wartości, umieszczone w rejestrach, które są jego argumentami.
- Wynik jest wartością 16-bitową i jest umieszczany w parze rejestrów `R1:R0`

Mnożenie całkowitoliczbowe

- Rozkaz `MUL`
  - mnoży liczby bez znaku,
  - wynik jest liczbą bez znaku,
  - argumenty mogą być w dowolnych rejestrach.
- Rozkaz `MULS`
  - mnoży liczby ze znakiem (U2),
  - wynik jest liczbą ze znakiem (U2),
  - argumenty mogą być w rejestrach `R16`, ..., `R31`.
- Rozkaz `MULSU`
  - mnoży liczbę ze znakiem (pierwszy argument) i liczbę bez znaku (drugi argument),
  - wynik jest liczbą ze znakiem (U2),
  - argumenty mogą być w rejestrach `R16`, ..., `R23`.

Liczby ułamkowe

- W ograniczonym zakresie dostępne są operacje arytmetyczne na liczbach ułamkowych.
- Jeśli wartość 8-bitowa reprezentuje liczbę całkowitą x (bez znaku lub ze znakiem), to przy jej interpretacji jako liczba ułamkowa jest to liczba $x2^{-7}$.
- Podobnie dla wartości 16-bitowej, reprezentującej liczbę całkowitą x, jest to liczba $x2^{-15}$.
- Liczba ułamkowa bez znaku jest z przedziału $[0; 2)$.
- Liczba ułamkowa ze znakiem jest z przedziału $[-1; 1)$.
- Do dodawania i odejmowania liczb ułamkowych używa się tych samych rozkazów, co dla liczb całkowitych.
- Do mnożenia służą rozkazy `FMUL`, `FMULS` i `FMULSU`.

Rozkazy mnożenia liczb ułamkowych

- Argumenty mogą być w rejestrach `R16`, ..., `R23`.
- Rozkaz wykonuje tę samą operację, co odpowiedni rozkaz mnożenia całkowitoliczbowego.
- Wynik jest przesuwany o jeden bit w lewo.
- Ewentualne przpepełnienie, czyli najstarszy bit jest umieszczany w znaczniku przeniesienia `C`.
- Mnożenie -1 przez -1 rozkazem `FMULS` daje wynik -1.
- Przykład:

```
LDI   R16, Q7(-0.135)
LDI   R17, Q7(0.753)
FMULS R16, R17
```

Gdy nie ma rozkazu mnożenia

- Mikrokontrolery z serii tiny nie mają rozkazu mnożenia.
- Poniższy kod prawie dobrze emuluje rozkaz `MUL`.

```
.macro emul
        push    r2
        push    r4
        push    @0
        mov     r4, @1
        pop     r2
        rcall   mul_emulator
        pop     r4
        pop     r2
.endmacro
```

Procedura mnożąca

```
mul_emulator:
        push    r3
        clr     r0
        clr     r1
        clr     r3
        rjmp    mul_emulator_loop_condition
mul_emulator_loop:
        lsr     r4
        brcc    mul_emulator_no_addition
        add     r0, r2
        adc     r1, r3
mul_emulator_no_addition:
        lsl     r2
        rol     r3
mul_emulator_loop_condition:
        tst     r4
        brne    mul_emulator_loop
```

Procedura mnożąca, dokończenie

```
        pop     r3
        ret
```

Pozostałe rozkazy

- Opóźnienie o jeden takt zegara

  `NOP`
- Zerowanie układu strażnika (ang. *watchdog*)

  `WDR`
- Zaśnij

  `SLEEP`
- Pułapka używana przy debugowaniu w układzie (JTAG, dW)

  `BREAK`