Zajęcia 1 – Ustawianie portu wejścia/wyjścia - 15.03.2021

Piny mikrokontrolera mogą pracować jako wejście lub wyjście. Chcemy, aby do naszego mikrokontrolera była podłączona dioda, która będzie się zapalała i gasła. Pin, do którego będzie podłączona dioda, musi być wyjściem. Gdy DDxn jest równe 0 mamy wejście, a gdy jest równy 1 - wyjście. W naszym przypadku, aby była możliwość zapalania i gaszenia diody, musimy mieć na wybranym porcie przynajmniej jedną jedynkę.

Table 10-1. Port Pin Configurations

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Str. 56 https://ww1.microchip.com/downloads/en/DeviceDoc/doc8006.pdf

Zdecydowaliśmy się na port A, który ma 8 nóżek. Ustawiamy port **PA1** jako wyjście.

```
DDRA.1 = 1
```

Reszta jest dla nas nieistotna, więc możemy ustawić cały port A jako wyjście.

```
DDRA = 255
```

Możemy także znaleźć przypadek, w którym potrzebujemy mieć je naprzemiennie (raz wyjście, raz wejście). Używamy do tego celu przedrostka **&B** i piszemy **1** dla wyjścia oraz **0** dla wejścia. Posługujemy się zapisem binarnym od lewej.

```
DDRA = &B10101010
```

Aby uzyskać efekt zapalania i gaszenia diody posłużymy się pętlą. Najpierw zapalimy **PORTA1**, poczekamy, a potem go zgasimy. Cały schemat będzie się powtarzać w pętli.

Rozpoczęcie pętli:

```
do
```

Zapalenie **PORTA1**:

```
PORTA.1 = 1
```

Jeżeli wait jest w sekundach to zapisujemy go w ten sposób (tutaj wait na czas jednej sekundy):

```
wait 1
```

Jeżeli potrzebujemy **wait** w milisekundach to do zapisu **wait** dodajemy ‘ms’ (tutaj wait na 10 milisekund):

```
waitms 10
```

Zgaszenie **PORTA1**:

PORTA.1 = 0

Gdybyśmy tak zakończyli pętlę, to mielibyśmy cały czas zapaloną diodę, bo gasła by ona jedynie na jeden takt zegarowy. Jeżeli chcemy tego uniknąć, to musimy znowu posłużyć się komendą **wait** (tutaj czekamy 2 sekundy):

wait 2

Zakończenie pętli:

loop

end

Cały program prezentuje się następująco:

```
DDRA.1 = 1 'ustawienie portu PA1 na wyjscie
DDRA = 255 'ustawienie calego portu A jako wyjscie
DDRA = &B10101010 'tam gdzie jest 1 jest wyjscie/ zapis binarny od lewej do prawej

do
PORTA.1 = 1
wait 1 'wait 1 sekunda
waitms 10 'wait 10 ms
PORTA.1 = 0
wait 2
loop
end
```

Zgodnie z powyższymi instrukcjami program ten pozwoli nam na zapalanie i gaszenie diody w ustalonych przez komendę wait odstępach czasowych z pomocą pętli.

Zajęcia 2 – Wywoływanie przerwania - 22.03.2021

Przerwanie jest zdarzeniem, które przerywa wykonywanie programu głównego i uruchamia specjalną funkcję obsługi przerwania. Gdy funkcja obsłuży przerwanie, następuje powrót do przerwanego programu i wznowienie jego wykonywania od miejsca, w którym został przerwany. Pin, do którego będzie podłączona dioda musi być ustawiony jako wejście – jako tenże pin wybraliśmy pin B2.

Ustawiamy pin B2 (ten, który obsługuje INT0) jako wejście, a pozostałe 3 piny (PB0, PB1 i PB3) jako wyjścia. &B oznacza zapis binarny.

DDRB = &B1011

Table 10-1. Port Pin Configurations

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Str. 56 https://ww1.microchip.com/downloads/en/DeviceDoc/doc8006.pdf

Następnie, zgodnie z powyższą tabelką, ustawiamy wszystkie bity portu B jako stan wysoki (wszystkie bity jako 1).

PORTB - &B1111

Mamy możliwość odbierania sygnału. Chcemy mieć przerwanie, aby w momencie zmiany stanu na nóżce, mikrokontroler wykonał jakiś program. Chcemy je uzyskać, kiedy jedna dioda minie drugą diodę – sygnał na wejściu sterownika zmienia swój stan z 1 na 0 dla zbocza opadającego, a więc dopiero, kiedy wartość na wejściu spadnie, dochodzi do przerwania.

Table 9-2. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Str. 51 https://ww1.microchip.com/downloads/en/DeviceDoc/doc8006.pdf

Jak widać w powyższej tabelce ISC01 musi mieć ustawioną 1, a ISC00 – 0. Zarówno dla ISC00, jak i dla ISC01, wartość domyślna to 0, więc musimy ustawić 1 do komórki ISC01.

9.3.1 MCUCR – MCU Control Register

The External Interrupt Control Register A contains control bits for interrupt sense control.

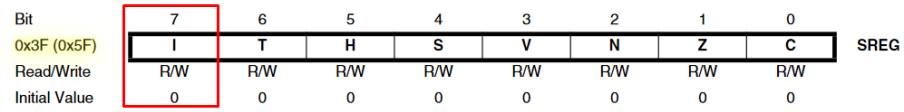
Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	BODS	PUD	SE	SM1	SM0	BODSE	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Str. 51 https://ww1.microchip.com/downloads/en/DeviceDoc/doc8006.pdf

Ustawiamy ISC01 (zgodnie z powyższą tabelą bit 1) w rejestrze MCUCR jako 1. Dla pozostałych wartości domyślnie nie ulegają zmianie – wszystkie są równe 0.

MCUCR.1 = 1

4.3.1 SREG – AVR Status Register



Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – I: Global Interrupt Enable**

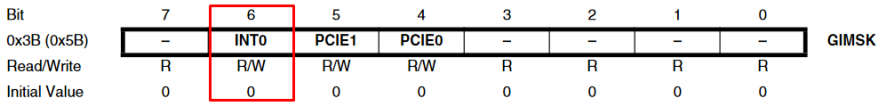
The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

Str. 9 https://ww1.microchip.com/downloads/en/DeviceDoc/doc8006.pdf

Następnie trzeba włączyć zezwolenie na wykonywanie przerwań w rejestrze SREG – zgodnie z powyższą tabelą ustawiamy 7 bit rejestru SREG jako 1. Bit 7 musi być ustawiony jako 1, żeby mikrokontroler w ogóle zezwalał na przerwanie:

SREG.7 = 1

9.3.2 GIMSK – General Interrupt Mask Register



Bit	7	6	5	4	3	2	1	0	
0x3B (0x5B)	–	INT0	PCIE1	PCIE0	–	–	–	–	GIMSK
Read/Write	R	R/W	R/W	R/W	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7, 3:0 – Res: Reserved Bits**

These bits are reserved in the ATtiny24/44/84 and will always read as zero.

- **Bit 6 – INT0: External Interrupt Request 0 Enable**

When the INT0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control bits (ISC01 and ISC00) in the External Interrupt Control Register A (EICRA) define whether the external interrupt is activated on rising and/or falling edge of the INT0 pin or level sensed. Activity on the pin will cause an interrupt request even if INT0 is configured as an output. The corresponding interrupt of External Interrupt Request 0 is executed from the INT0 Interrupt Vector.

Str. 51 https://ww1.microchip.com/downloads/en/DeviceDoc/doc8006.pdf

Kolejny krok to odpowiednie ustawienie w rejestrze GIMSK. Zgodnie z powyższą tabelą ustawiamy bit 6 (odpowiadający INT0) jako 1, co pozwala na wykonanie zewnętrznego przerwania INT0.

GIMSK.6 = 1

Następnie deklarujemy zmienną A typu bajt.

DIM A as byte

9.1 Interrupt Vectors

The interrupt vectors of ATtiny24/44/84 are described in Table 9-1 below.

Table 9-1. Reset and Interrupt Vectors

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset
2	0x0001	INT0	External Interrupt Request 0
3	0x0002	PCINT0	Pin Change Interrupt Request 0
4	0x0003	PCINT1	Pin Change Interrupt Request 1

Str. 48 https://ww1.microchip.com/downloads/en/DeviceDoc/doc8006.pdf

Oraz ustalamy, co stanie się w momencie przerwania (INT0 - External Interrupt Request0, w wolnym tłumaczeniu zewnętrzne żądanie przerwania 0, który, zgodnie z powyższą tabelą, ma najwyższy priorytet zaraz po RESET) – program przejdzie do ‘pisz’.

ON INT0 pisz

Następnie ustawiamy pętle nieskończoną. Jedynie przerwanie INT0 powoduje wyjście z tejże pętli.

do

Zakończenie pętli:

loop

end

Kolejne linijki to instrukcja działania ‘pisz’ – w momencie przerwania program zwiększy wartość zmiennej A o 1.

pisz:

INCR A

Zakończenie podprogramu zajmującego się obsługą przerwania:

return

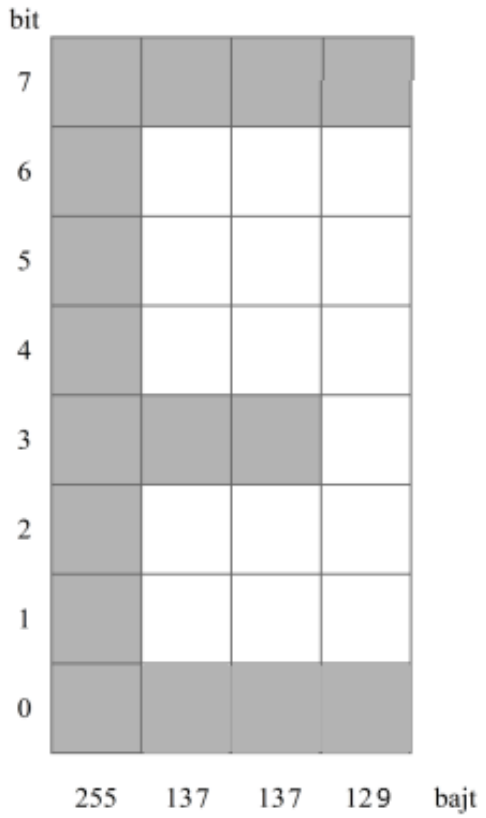
Cały program prezentuje się następująco:

```
DDRB = &B1011 'ustawienie portu PB2 jako wejscie,a PB1, 3 i 4 jako wyjscie; DDRB - ustawiamy cos w B
PORTB = &B1111 'ustawienie wsszystkich bitów na stan wysoki
MCUCR.1 = 1 'ustawiamy ISC01 jako 1, reszta pozostaje jako  0 - ustawilismy przerwanie INT0 na zbocze opadajace
SREG.7 = 1 'zezwolenie na wykonywanie przerwan - ustawienie 7 bitu rejestru SREG jako 1
GIMSK.6 = 1 'ustawiamy 6 bit rejestru GIMSK jako 1, co pozwala nam na wykonanie zewnetrznego przerwania INT0

DIM A as byte 'zadeklarowanie zmiennej A typu bajt

ON INT0 pisz

do 'petla nieskonczona; jedynie przerwanie INT0 powoduje wyjscie z niej

loop

end

pisz: 'pisz zwieksza wartosc A
INCR A
return
```

Tak więc program ten, po włączeniu odpowiednich bitów w rejestrach MCUCR, SREG oraz GIMSK, obsługuje przerwanie, zwiększając wartość zmiennej A o 1 przy każdym przerwaniu INT0.

Zajęcia 3 – Program wyświetlający literę - 29.03.2021

Poniżej przedstawiony pomocniczy projekt litery “E”:



W pamięci mikrokontrolera taka litera “E”, będzie reprezentowana przez następujący ciąg bajtów: 255, 137, 137, 129.

Każda komórka to liczba “2” podniesiona do odpowiedniej potęgi (którą widać po lewej stronie obrazka). Następnie sumujemy wszystkie wartości w danej kolumnie (pusta komórka jest zerem).

Korzystając z wiedzy nabytej na poprzednich zajęć i projektu litery można przystąpić do pisania programu:

DDRA = &B11111111

Ustawiamy cały Port A jako wyjści.

DDRB = &B1011

Ustawiamy port PB2 jako wejście oraz porty: PB1, PB3 i PB4 jako wyjście.

PORTB = &B1111

Ustawiamy wszystkie bity jako stan wysoki.

MCUCR.1 = 1

Ustawiamy ISC01 (zgodnie z punktem 9.3.1 str. 51 dokumentacji) w rejestrze MCUCR jako 1.

GIMSK.6 = 1

Zgodnie z punktem 9.3.2 str. 51 dokumentacji ustawiamy bit 6 (odpowiadający za INT0) jako 1. Czyli pozwalamy na wykonanie zewnętrznego przerwania INT0.

SREG.7 = 1

Włączamy zezwolenie na wykonywanie przerwań w rejestrze SREG (zgodnie z punktem 4.3.1 str. 9 dokumentacji ustawiamy 7 bit rejestru SREG jako 1).

DIM A1 as byte

Deklarujemy zmienne jako typ bajt.

A1 = &B11111111

Nadajemy zmiennym wartości odpowiadające liczbom naszej zaprojektowanej litery (w systemie binarnym).

ON INT0 wyswietl

Ustalamy, co stanie się w momencie przerwania – program przejdzie do ‘wyswietl’.

do

Następnie ustawiamy pętle nieskończoną. Jedynie przerwanie INT0 powoduje wyjście z tejże pętli.

loop

end

Ustalamy zakończenie pętli.

PORTA = A1

Ustawiamy porty jako wcześniej zadeklarowane zmienne.

Waitus 469

Ustawiamy przerwanie, w którym wyświetlamy pierwszy segment, który już zakodowaliśmy. Będzie ona trwałła 469 mikrosekund co jest wartością 1/128 pełnego obrotu, przy 1000 obrotów na minutę).

Następnie dla wszystkich segmentów powtarzamy zakodowanie portu oraz przerwania.

return

Kończymy program.

```
DDRA = &B11111111 'ustawienie calego portu A jako wyjscie
DDRB = &B1011 'ustawienie portu PB2 jako wejscie,a PB1, 3 i 4 jako wyjscie; DDRB - ustawiamy cos w B
PORTB = &B1111 'ustawienie wszystkich bitów na stan wysoki
MCUCR.1 = 1 'ustawiamy ISC01 jako 1, reszta pozostaje jako  0 - ustawilismy przerwanie INT0 na zbocze opadajace
SREG.7 = 1 'zezwolenie na wykonywanie przerwan - ustawienie 7 bitu rejestru SREG jako 1
GIMSK.6 = 1 'ustawiamy 6 bit rejestru GIMSK jako 1, co pozwala nam na wykonanie zewnetrznego przerwania INT0

DIM A1 as byte 'zadeklarowanie zmiennej A1 typu bajt
DIM A2 as byte 'zadeklarowanie zmiennej A2 typu bajt
DIM A3 as byte 'zadeklarowanie zmiennej A3 typu bajt
DIM A4 as byte 'zadeklarowanie zmiennej A4 typu bajt

A1 = &B11111111 'nadanie zmiennej A1 wartosci 255 w systemie binarnym
A2 = &B10001001 'nadanie zmiennej A2 wartosci 137 w systemie binarnym
A3 = &B10001001 'nadanie zmiennej A3 wartosci 137 w systemie binarnym
A4 = &B10000001 'nadanie zmiennej A4 wartosci 129 w systemie binarnym

ON INT0 wyswietl

do 'petla nieskonczona; jedynie przerwanie INT0 powoduje wyjscie z niej

loop

end

wyswietl: 'wyswietl zmienia wartosc PORTA na wartosci zmiennych odpowiadajacych kolumnom

PORTA = A1 'ustawiamy wartosc PORTA na wartosc zmiennej A1
waitus 469 'waitus - czekanie w mikrosekundach - czekamy 469 mikrosekund
PORTA = A2
waitus 469
PORTA = A3
waitus 469
PORTA = A4
waitus 469
PORTA = 0

return
```

Podsumowanie:

Program zgodnie z przeznaczeniem odpowiada za wyświetlanie nam litery “E”.

Zajęcia 4 – Działanie na licznikach mikrokontrolera - 12.04.2021

Jeśli do wyświetlania litery używamy komendy **wait** jak w przypadku poprzednich zajęć to mikrokontroler zużywa całą moc na liczenie czasu i nie może zrobić nic innego. Aby tego uniknąć na stanowisku z artykułu http://www.komel.katowice.pl/ZRODLA/FULL/119/ref_26.pdf wewnętrzne liczniki mikrokontrolera odmierzają czas. Jeden z tych liczników odmierza czas pełnego obrotu tarczy. Drugi, który ma mniejszą pojemność odmierza 1/128 pełnego obrotu. W tym dużym liczniku (16-bitowym), w którym odmierzamy pełny obrót, zostało to podzielone na 2^n aby można było łatwo wyznaczyć czas trwania jednego wycinka. W licznikach w odpowiednich rejestrach trzeba poustawiać jaki ma być **prescaler** tego licznika i jak chcemy go obsługiwać.

Prescaler – dzielnik dzielący przed jakąś czynnością.

Zaczynamy od licznika 16-bitowego (odpowiedzialnego za odmierzanie czasu trwania pełnego obrotu) aby później dzieląc to przez 2^7 wyznaczyć czas trwania jednej sekwencji sterowania.

Wiemy, że nasz mikrokontroler posiada nominalną częstotliwość 8mHz (można go ustawić na 4mHz; str. 27 dokumentacji).

Table 6-4. Internal Calibrated RC Oscillator Operating Modes

CKSEL3:0	Nominal Frequency
0010 (1)	8.0 MHz

I taka właśnie częstotliwość 8 MHz może być dostarczana do licznika 16-bitowego lub można tę częstotliwość podzielić. Czyli to co dostarczamy do licznika z generatora wewnętrznego możemy podzielić przez 1 lub 8 lub 64 lub 256 lub 1024. **Prescaler** dobieramy tak aby w czasie trwania jednego obrotu licznik naliczył jak najwięcej, ale żeby się nie przepełnił (gdy licznik się przepełnia to liczy znowu od zera; str. 111 dokumentacji).

Table 12-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Gdybyśmy mieli prescaler równy 1 to w trakcie jednego obrotu licznik naszego mikrokontrolera taktowany 8mHz dla 1 obr./s naliczyłby 8 000 000 (bo 8MHz to 8 000 000 Hz i 8 000 000 Hz razy 1 s).

Gdybyśmy mieli prescaler równy 1 to w trakcie jednego obrotu licznik naszego mikrokontrolera taktowany 8mHz dla obrotu trwającego 0,1 sekundy naliczyłby 800 000 (bo 8mHz to 8 000 000 Hz i 8 000 000 Hz razy 0,1 s).

Tak więc jeżeli nasz obrót trwa 0,06 sekundy to dla prescalera równego 1 naliczy 480 000 (bo 8MHz to 8 000 000 Hz i 8 000 000 Hz razy 0,06 s).

Liczba 16-bitowa to 65 636. Więc dla prescalera równego 1, skoro nasz licznik naliczy 480 000 przekręci się to 7 razy. Z tego względu musimy wybrać większy prescaler.

Dla prescalera równego 8 mamy już liczbę 60 000 (480 000/8). Ten prescaler jest poprawny, jednak dla większego bezpieczeństwa można wybrać prescaler równy 64.

Decydujemy się na wybranie prescalera równego 8. Jak przedstawia powyższa tabela (12-6.) musimy ustawić CS11 na 1.

12.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Jak przedstawia powyższa tabela (str. 110 dokumentacji) aby ustawić CS11 na 1 musimy ustawić pierwszy bit rejestru TCCR1B na 1:

TCCR1B.1 = 1

Teraz dalsza część programu będzie odpowiadała za to aby gdy wystapi przerwanie (fotokomórka zostaje oświetlona) złapać wartość naszego licznika i zapisać do zmiennej a potem wyzerować. Następnie na przechwyconej liczbie możemy realizować jakieś działania.

Na początku musimy ustawić odpowiednie porty na wejście i wyjście oraz musi być aktywne przerwanie INT0. Możemy posłużyć się kodem z poprzedniego programu (zajęcia 2):

Ustawiamy cały port A jako wyjście:

DDRA = &B11111111

Ustawiamy pin B2 (ten, który obsługuje INT0) jako wejście, a pozostałe 3 piny (PB0, PB1 i PB3) jako wyjścia:

```
DDRB = &B1011
```

Ustawiamy wszystkie bity portu B jako stan wysoki:

```
PORTB - &B1111
```

Ustawiamy ISC01 (zgodnie z punktem 9.3.1 str. 51 dokumentacji) w rejestrze MCUCR jako 1:

```
MCUCR.1 = 1
```

Włączamy zezwolenie na wykonywanie przerwań w rejestrze SREG (zgodnie z punktem 4.3.1 str. 9 dokumentacji ustawiamy 7 bit rejestru SREG jako 1):

```
SREG.7 = 1
```

Zgodnie z punktem 9.3.2 str. 51 dokumentacji ustawiamy bit 6 (odpowiadający za INT0) jako 1. Czyli pozwalamy na wykonanie zewnętrznego przerwania INT0:

```
GIMSK.6 = 1
```

Następnie deklarujemy dwie zmienne typu bajt i jedną zmienną typu word (16-bitowa, dodatnia liczba), które przydadzą nam się w późniejszej części:

```
DIM L as byte

DIM H as byte

DIM D as word
```

Dodajemy przerwanie. Czyli decydujemy co program ma zrobić, gdy dojdzie do przerwania:

```
ON INT0 wyswietl
```

Dodajemy pętlę, z której wychodzimy w momencie przerwania INT0:

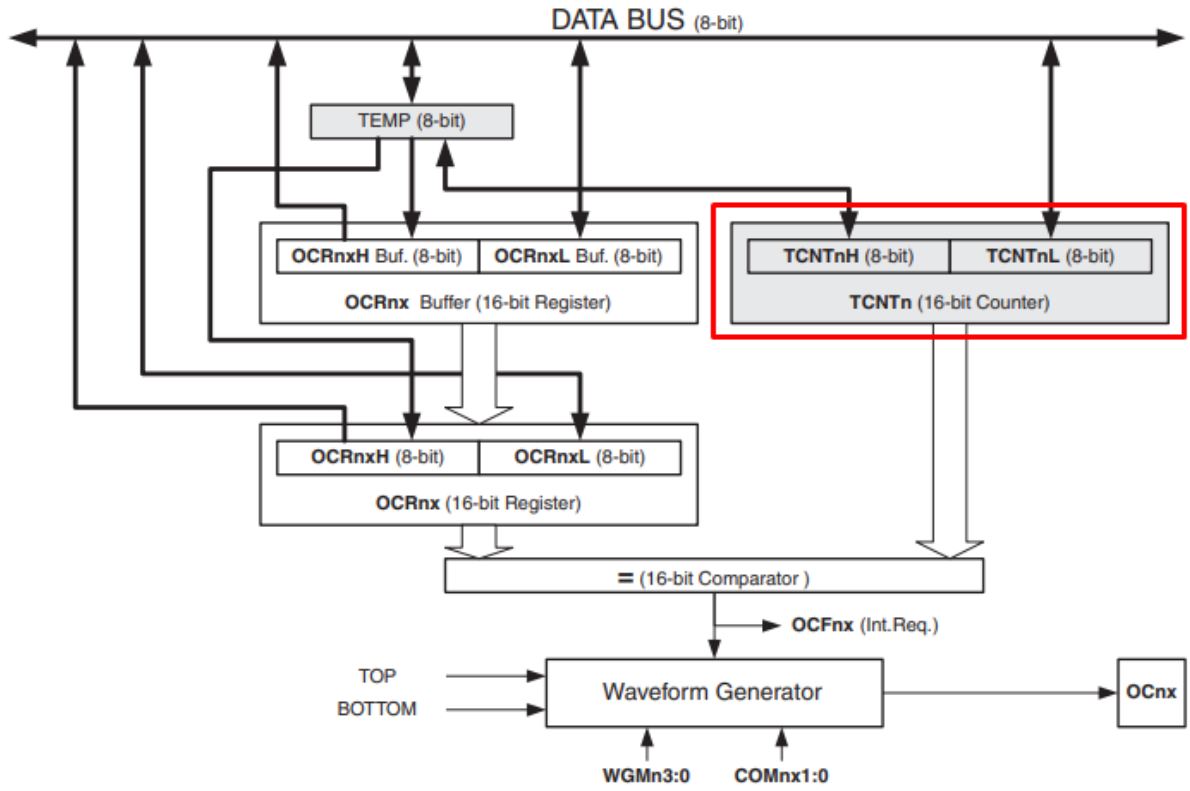
```
do

loop

end
```

Następnie określamy przebieg programu w momencie przerwania.

```
wyswietl:
```

Na schemacie licznika ze strony 93 dokumentacji mamy przedstawione rejestry TCNTnH i TCNTnL (każdy po 8 bitów). Rejestry są szerzej opisane w punkcie 12.11.4 dokumentacji. W tych rejestrach zliczane są impulsy. Jeżeli my taktujemy nasz licznik to tam zwiększa się wartość. Nasza liczba 16-bitowa ma 8 bitów w rejestrze TCNT1H i kolejne 8 mniej ważnych w TCNT1L.

Figure 12-4. Output Compare Unit, Block Diagram



Tak więc najpierw zapisujemy liczby w zmiennych by później móc odczytać określoną wartość (zgodnie z dokumentacją najpierw zapisujemy L a później H z tego względu, że gdybyśmy odczytali najpierw H to mogłoby się już zmienić L):

L = TCNT1L

H = TCNT1H

Zerujemy rejestry, aby nie wpływały na nasze dokładności obliczeń:

TCNT1L = 0

TCNT1H = 0

Tworzymy 16-bitową zmienną D zapisaną za pomocą L i H (pamiętając, że możemy wykonać tylko jedną operację w jednej linii):

D = H*256

D = D + L

Na koniec, abyśmy widzieli do ilu nalicza nasz licznik wyświetlamy 8 najważniejszych bitów:

PORTA = H

Kończymy program:

return

Aby podzielić ten jeden obrót na 128 części wystarczy naszą zmienną D podzielić przez 2^7.

Gotowy program prezentuje się następująco:

```
TCCR1B.1 = 1 'ustawienie pierwszego bitu rejestru TCCR1B na 1

DDRA = &B11111111 'ustawienie calego portu A jako wyjscie
DDRB = &B1011 'ustawienie portu PB2 jako wejscie,a PB1, 3 i 4 jako wyjscie; DDRB - ustawiamy cos w B
PORTB = &B1111 'ustawienie wszystkich bitów na stan wysoki
MCUCR.1 = 1 'ustawiamy ISC01 jako 1, reszta pozostaje jako  0 - ustawilismy przerwanie INT0 na zbocze opadajace
SREG.7 = 1 'zezwolenie na wykonywanie przerwan - ustawienie 7 bitu rejestru SREG jako 1
GIMSK.6 = 1 'ustawiamy 6 bit rejestru GIMSK jako 1, co pozwala nam na wykonanie zewnetrznego przerwania INT0

DIM L as byte 'zadeklarowanie zmiennej L typu bajt
DIM H as byte 'zadeklarowanie zmiennej H typu bajt
DIM D as word 'zadeklarowanie zmiennej D typu word

ON INT0 wyswietl

do 'petla nieskonczona; jedynie przerwanie INT0 powoduje wyjscie z niej

loop

end

wyswietl: 'wyswietl zmienia wartosc PORTA na wartosci zmiennych odpowiadajacych kolumnom

L = TCNT1L 'zapisywanie licznika do zmiennej L
H = TCNT1H 'zapisywanie licznika do zmiennej H

TCNT1L = 0 'zerowanie TCNT1L
TCNT1H = 0 'zerowanie TCNT1H

D = H*256 'zapisywanie H do zmiennej D
D = D+L 'zapisywanie L do zmiennej D

PORTA = H 'wyswietlanie najwazniejszych 8 bitow

return
```

Tak więc, przy pomocy takiego programu możemy zaprogramować mikrokontroler, aby sprzętowo mierzyć czas jednego obrotu.

Zajęcia 5 – Działanie na licznikach mikrokontrolera - ciąg dalszy - 19.04.2021

Na dzisiejszych zajęciach zajmiemy się ‘małym’, 8-bitowym licznikiem, analogicznie do 16-bitowego licznika z poprzednich zajęć, z tym, że chcemy uzyskać przerwanie od licznika 8-bitowego, aby kiedy mając wyznaczony czas jednego pełnego obrotu, podzielony na 128 części, co jedną część tegoż obrotu nasz 8-bitowy licznik mógł zgłosić przerwanie, że należy zastosować kolejne wysterowanie diod. Gdy licznik 8-bitowy doliczy do wartości, jaka została od niego wpisana, to zgłosi nam przerwaniu. W tym przerwaniu zasterujemy A od itego wycinka. Na podstawie dokumentacji 8-bitowego licznika musimy powstawiać odpowiednie rejestry, żeby tenże licznik mógł poprawnie pracować i zgłaszać przerwania.

Fragmenty kodu omówione na wcześniejszych zajęciach zostaną pominięte w części omawiania kodu.

Aby licznik mógł poprawnie działać, musimy ustawić odpowiednie wartości w rejestrze TCCR0B. Zawartość rejestru TCCR0B:

Bit	7	6	5	4	3	2	1	0	
0x33 (0x53)	**FOC0A**	**FOC0B**	–	–	**WGM02**	**CS02**	**CS01**	**CS00**	**TCCR0B**
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Str. 83 https://ww1.microchip.com/downloads/en/DeviceDoc/doc8006.pdf

Użyjemy prescalera 64, więc zgodnie z poniższą i powyższą tabelą, musimy ustawić odpowiednie bity rejestru TCCR0B.

Table 11-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Str. 84 https://ww1.microchip.com/downloads/en/DeviceDoc/doc8006.pdf

Ustawienie zerowego bitu rejestru TCCR0B na 1:

TCCR0B.0 = 1

Ustawienie pierwszego bitu rejestru TCCR0B na 1:

TCCR0B.1 = 1

Musimy również dokonać zmian w rejestrze TIMSK0, aby włączyć obsługę przerwania przy przepełnieniu licznika. Zgodnie z poniższą tabelą musimy zmienić pierwszy bit rejestru TIMSK0.

Bit	7	6	5	4	3	2	1	0	
0x39 (0x59)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Str. 85 https://ww1.microchip.com/downloads/en/DeviceDoc/doc8006.pdf

Zgodnie z dokumentacją, TOIE0 odpowiada za włączenie obsługi przerwania przy przepełnieniu:

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

Str. 85 https://ww1.microchip.com/downloads/en/DeviceDoc/doc8006.pdf

Ustawienie pierwszego bitu rejestru TIMSK0 jako 1:

TIMSK0.1 = 1

Następnie deklarujemy zmienną ‘I’ typu bajt, która będzie służyć jako licznik:

DIM I as byte

Kolejny krok to deklaracja zmiennej A(4) typu bajt, która będzie miała 4 ‘podzmienne’ A(1), A(2), A(3), A(4), przez które w dalszej części programu będzie ‘przechodzić’ **IF**:

DIM A(4) as byte

Nadanie kolejny ‘podzmiennym’ odpowiednich wartości (A(1) = 255, A(2) = 137, A(3) = 137 oraz A(4) = 129; wszystkie te wartości w systemie binarnym):

A(1) = &B11111111

A(2) = &B10001001

A(3) = &B10001001

A(4) = &B10000001

Ustalamy, co stanie się w momencie przerwania (OC0A – Output Compare Match output) – program przejdzie do ‘pasek’:

ON OC0A pasek

Ponieważ przy 16-bitowym liczniku użyliśmy prescalera 8, to musimy podzielić D przez 1024, bo 8*128 = 1024, gdzie 128 to liczba podziału jednego pełnego obrotu.

D = D/1024

Następnie dokonujemy zapisania niższych bitów zmiennej D do rejestru OCR0A, który, jak wcześniej ustaliliśmy, odpowiada za porównywanie z wartością licznika TCNT0:

OCR0A = Low(D)

Oraz ustawiamy wartość zmiennej I jako 1:

I = 1

Kolejne linijki to instrukcja działania ‘pasek’:

pasek:

Jeśli ‘I’ jest mniejsze niż 5, ustawiamy PORTA jako A(I):

IF I < 5 THEN

    ## PORTA = A(I)

Jeśli i jest większe lub równe 5, ustawiamy PORTA jako 0:

ELSE

    ## PORTA = 0

Zakończenie instrukcji **IF**:

END IF

Zwiększenie wartości zmiennej I o 1:

INCR I

Zakończenie programu:

return

```
TCCR1B.1 = 1 'ustawienie pierwszego bitu rejestru TCCR1B na 1
TCCR0B.0 = 1 'ustawienie zerowegoo bitu rejestru TCCR0B na 1
TCCR0B.1 = 1 'ustawienie pierwszego bitu rejestru TCCR0B na 1

DDRA = &B11111111 'ustawienie calego portu A jako wyjscie
DDRB = &B1011 'ustawienie portu PB2 jako wejscie,a PB1, 3 i 4 jako wyjscie; DDRB - ustawiamy cos w B
PORTB = &B1111 'ustawienie wszystkich bitów na stan wysoki
MCUCR.1 = 1 'ustawiamy ISC01 jako 1, reszta pozostaje jako  0 - ustawilismy przerwanie INT0 na zbocze opadajace
SREG.7 = 1 'zezwolenie na wykonywanie przerwan - ustawienie 7 bitu rejestru SREG jako 1
GIMSK.6 = 1 'ustawiamy 6 bit rejestru GIMSK jako 1, co pozwala nam na wykonanie zewnetrznego przerwania INT0
TIMSK0.1 = 1 'ustawienie pierwszego bitu rejestru TIMSK0 jako 1

DIM L as byte 'zadeklarowanie zmiennej L typu bajt
DIM H as byte 'zadeklarowanie zmiennej H typu bajt
DIM D as word 'zadeklarowanie zmiennej D typu word
DIM I as byte 'zadeklarowanie zmiennej I typu bajt, ktora bedzie sluzyc jako licznik

DIM A(4) as byte 'zadeklarowanie zmiennej A1 typu bajt

A(1) = &B11111111 'nadanie zmiennej A1 wartosci 255 w systemie binarnym
A(2) = &B10001001 'nadanie zmiennej A2 wartosci 137 w systemie binarnym
A(3) = &B10001001 'nadanie zmiennej A3 wartosci 137 w systemie binarnym
A(4) = &B10000001 'nadanie zmiennej A4 wartosci 129 w systemie binarnym

ON INT0 wyswietl
ON OC0A pasek

do 'petla nieskonczona; jedynie przerwanie powoduje wyjscie z niej

loop

end

wyswietl: 'etykieta wyswietl

L = TCNT1L 'zapisywanie licznika do zmiennej L
H = TCNT1H 'zapisywanie licznika do zmiennej H

TCNT1L = 0 'zerowanie TCNT1L
TCNT1H = 0 'zerowanie TCNT1H

D = H*256 'zapisywanie H do zmiennej D
D = D+L 'zapisywanie L do zmiennej D

D = D/1024 'podzielenie D przez 1024, bo 8*128 = 1024
OCR0A = Low(D) 'zapisanie nizszych bitów zmiennej D do rejestru OCR0A, który odpowiada za porównywanie z wartoscia licznika TCNT0

I = 1 'ustawienie wartosci zmiennej I jako 1

return

pasek: 'etykieta pasek i obsluga przerwania OC0A

IF I < 5 THEN 'jesli i jest mniejsze niz 5 to ustawiamy PORTA jako A(I)
   PORTA = A(I)

ELSE
   PORTA = 0 'jesli i jest wieksze lub rowne 5 to ustawiamy PORTA jako 0

END IF 'zakonczenie ifa

INCR I 'zwiekszenie wartosci zmiennej I o 1

return
```

Tak więc, przy pomocy takiego programu możemy zaprogramować mikrokontroler, aby sprzętowo mierzył on czas jednego obrotu oraz sterował diodami.

Zajęcia 6 – Pisanie własnego prostego programu- 26.04.2021

Prawie całe zajęcia poświęciliśmy na samodzielnym pisaniu programu, który odpowiadał za miganie poszczególnymi diodami.

W naszym programie zdecydowaliśmy się na port A, który ma 8 nóżek.

DDRA = &B10101010

Powyższa komenda pozwala nam na ustawianie w sposób binarny wejść i wyjść przy pomocy przedrostka **&B.** Dla wyjścia piszemy **1** oraz dla wejścia **0**.

do

Rozpoczęcie pętli:

PORTA.1 = 1

Zapalenie **PORTA1**:

waitms 500

wait trwający 500 milisekund

PORTA.1 = 0

Zgaszenie **PORTA1**:

W podobny sposób dodajemy kolejne diody w portach 3, 5 i 7, które zaświecamy i gasimy.

loop

end

Zakończenie pętli:

```
DDRA = &B10101010

do

PORTA.1 = 1
waitms 500
PORTA.1 = 0
waitms 500

PORTA.1 = 1
PORTA.3 = 1
waitms 500
PORTA.1 = 0
PORTA.3 = 0
waitms 500

PORTA.1 = 1
PORTA.3 = 1
PORTA.5 = 1
waitms 500
PORTA.1 = 0
PORTA.3 = 0
PORTA.5 = 0
waitms 500

PORTA.1 = 1
PORTA.3 = 1
PORTA.5 = 1
PORTA.7 = 1
waitms 500
PORTA.1 = 0
PORTA.3 = 0
PORTA.5 = 0
PORTA.7 = 0
waitms 500

loop

end
```

Podsumowując wcześniejsze zajęcia pozwoliły nam na samodzielne napisanie programu, który później został zaprezentowany przez prowadzącego przedmiot na stanowisku. Diody poprawnie oświecały się i gasiły tworząc oczekiwany efekt.