Techniki Mikroprocesorowe - Projekt nr 2.

Adam Kowalewski, Maciej Kłos

21 kwietnia 2016

Zaprojektować licznik Johnsona 4-ro bitowy z funkcją asynchronicznego ładowania z przełączników szesnastkowych oraz funkcją inkrementowania reagującą zboczem. Obie czynności mają być aktywowane przyciskami, przy czym ładowanie ma być wykonywane w przerwaniu, a oczekiwanie na naciśnięcie przycisku inkrementacji ma być wykonywane w pętli głównej.

Wymagane informacje

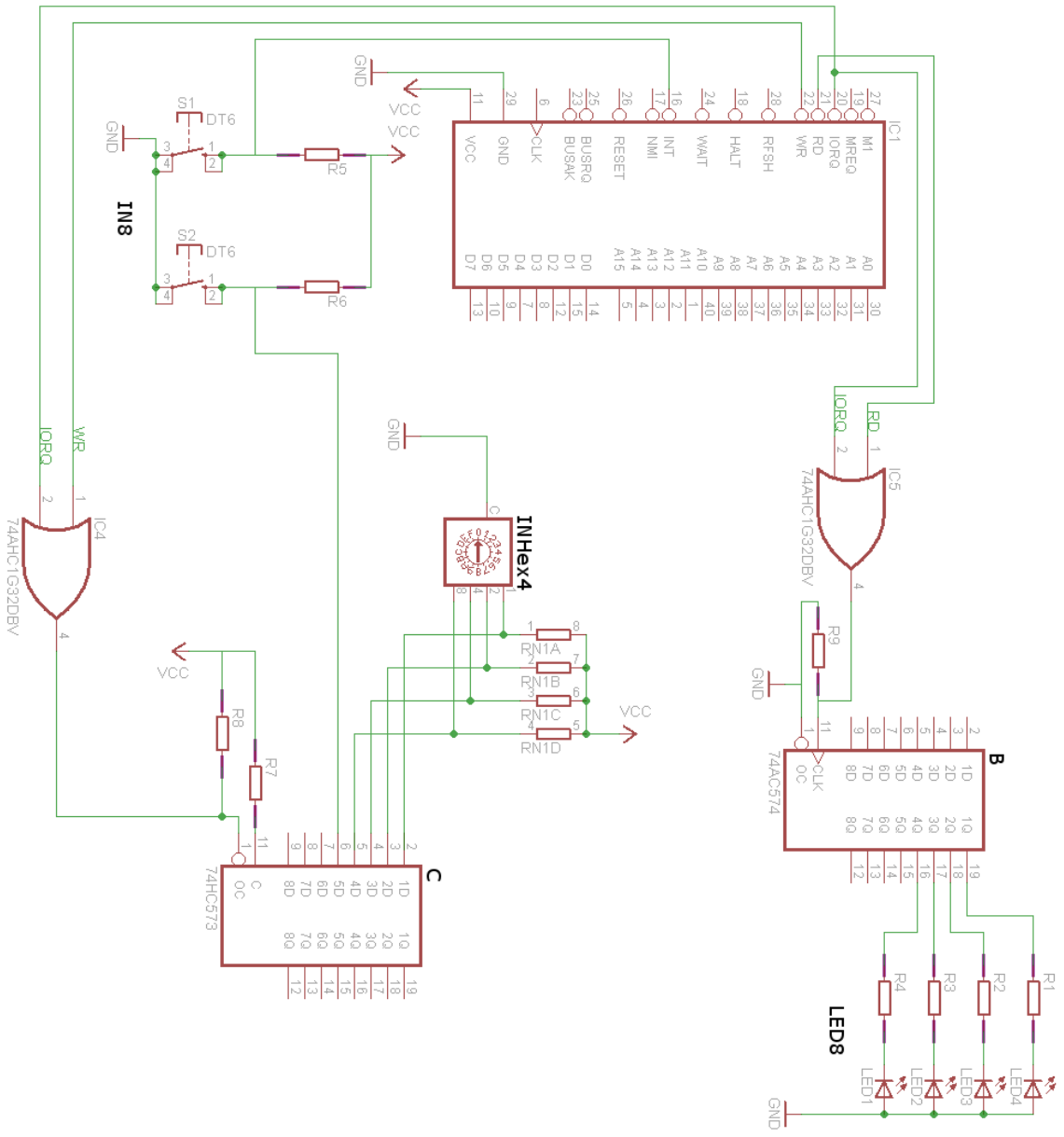
Licznik Johnsona, w przeciwieństwie do innych sposobów kodowania (NKB, Gray etc.) przy 4-ro bitowym zapisie nie wykorzystuje wszystkich 16 możliwości. Przy takim zapisie osiągamy maksymalnie 8 możliwości, które wypisane są poniżej :

Numer kodu	Wartość binarna kodu
0_{10}	0000_2
1_{10}	0001_2
2_{10}	0011_2
3_{10}	0111_2
4_{10}	1111_2
5_{10}	1110_2
6_{10}	1100_2
7_{10}	1000_2

Kod, jak widać, jest cykliczny, przy czym tylko określone wartości spośród wszystkich możliwych 4-ro bitowych można użyć. Inkrementowanie kodu Johnsona jest oparte na następującej zasadzie :

1. Przesuń arytmetycznie aktualną wartość kodową w lewo o jedną pozycję
2. Zapisz w pamięci bit, który jest tracony podczas przesuwania (najstarszy)
3. Jeśli bit ten jest równy zero, najmłodszy, pierwszy bit, ustaw na jeden
4. Jeśli natomiast jest równy jeden, najmłodszy ustaw na zero

Rozwiązanie zadania - strona sprzętowa



Rysunek 1: Schemat sprzętowy zaprojektowanego przez nas licznika

Rozwiązanie zadania - strona programowa

Reset programu

```
ORG 1800h
;--------------------------------------------------
; RESET, start programu spod adresu 0
;--------------------------------------------------
RESET:
  LD  SP, 0x2000 ; inicjuj stos

  IM  1 ; pierwszy typ przerwan

  ; inicjuj licznik
  LD  A, 0xF0
  LD  (ZMIENNA), A
  OUT (0x00), A ; wyswietl wynik na ledach

  LD  (ZMIENNA_INT), A ; zaladuj zmienna_int zerem

  EI
  JR  LOOP  ; skocz do petli glownej
```

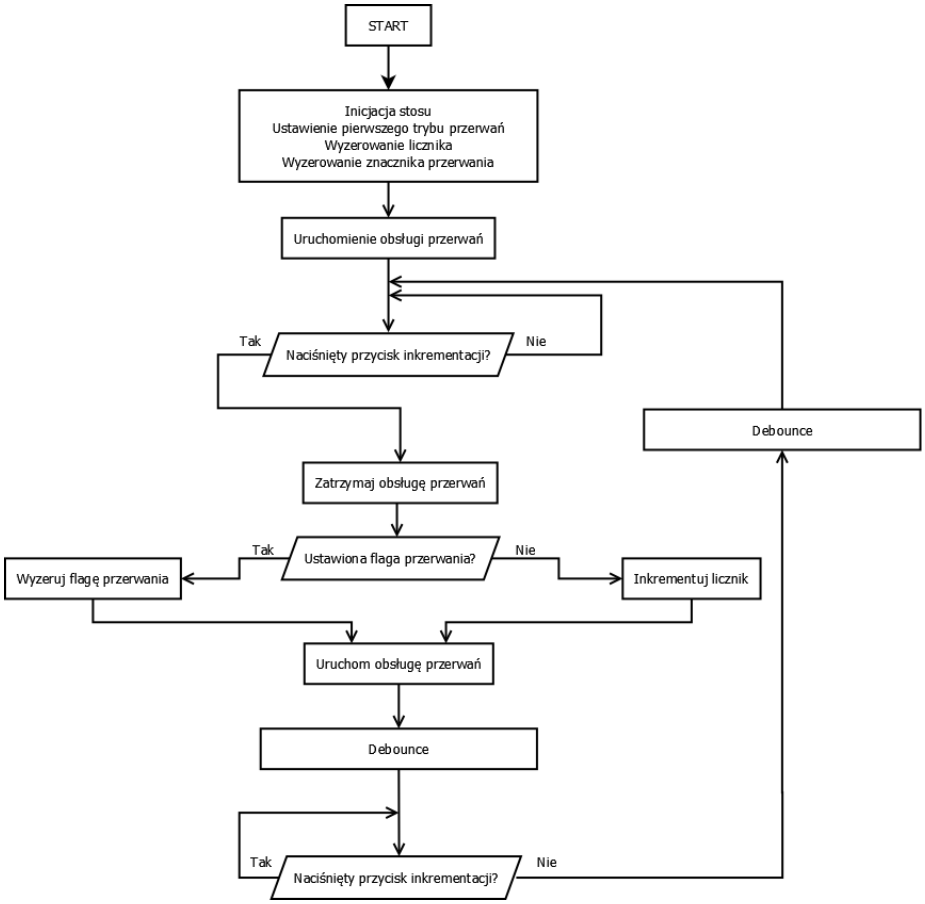
Procedura obsługi `RESETu`, tak jak zostało to przewidziane w procesorze Z80, jest umieszczona na samym początku pamięci (u nas ma to miejsce pod adresem `0x1800`, gdyż pamięć w używanym przez nas układzie emulującym Z80 jest przesunięta).

Pierw ustawiamy wskaźnik stosu na wartość `0x2000`. Dostępna dla nas pamięć kończy się właśnie w tym miejscu. Właściwie to ostatnia dostępna dla nas komórka znajduje się w miejscu `0x1999`, ale w związku z tym, iż wykonywana jest **predekrementacja**, to pierw zostanie zmniejszony wskaźnik stosu o jeden, a dopiero potem zapisana zostanie dana (w przypadku operacji `PUSH`).

Następnie inicjujemy zmienne używane przez nas w programie. Będą to zmienne *ZMIENNA* oraz *ZMIENNA_INT* . Obie inicjujemy **zerami**. Na koniec skaczemy do pętli głównej programu.

Pętla główna programu

Pierw przestawimy schemat blokowy, w jaki to sposób działa nasza pętla główna po wyjściu z RESETu :



Rysunek 2: Pętla główna

Teraz pora na kod, czyli jak to zaprogramowaliśmy (((oklaski)))

```
;-------------------------------------------------------
; Petla glowna programu
;-------------------------------------------------------
LOOP:
  OCZEKIWANIE:     ; aktywnie oczekujemy na nacisniecie
    IN  A, (00h)   ; pobierz stan hexa
    BIT INC_BTN, A       ; sprawdz, czy nacisnieto przycisk
    JR  NZ, OCZEKIWANIE ; jesli nie, powtorz

  ; przycisk nacisniety.
  ; pierw sprawdzamy, czy nie byly oba wcisniete
  DI

  LD    A, (ZMIENNA_INT) ; pobierz stan flagi
  BIT   FLAGA, A ; sprawdz stan

  JR    NZ, NOT_INC ; omijamy, jesli flaga ustawiona
  CALL  INKREMENTUJ ; jeśli nie, inkrementujemy licznik
  JR    AFTER_INC

NOT_INC:
  RES   FLAGA, A ; skasuj flage
  LD    (ZMIENNA_INT), A

AFTER_INC:
  EI

  CALL  DELAY ; Odczekujemy debouncing

  OCZEKIWANIE2: ; oczekiwanie, aby przycisk zostal puszczony
    IN  A, (0x00) ; zaladuj stan hexa
    BIT INC_BTN, A ; sprawdz stan przycisku
    JR  Z, OCZEKIWANIE2 ; jesli nadal jest wcisniety, petla

  CALL  DELAY ; przycisk puszczony, odczekaj

  JR  LOOP ; zapetl program
```

Pętla główna nie jest skomplikowana. Jedyne zadanie jakie jest w niej do wykonania to aktywne oczekiwanie na naciśnięcie przycisku inkrementacji.

Za oczekiwanie na naciśnięcie przycisku odpowiada pierwsza pod-pętla o etykiecie *OCZEKIWANIE*. W niej po prostu cały czas czytamy port (pod adresem `0x00`, właściwie to nie możemy tu wstawić dowolną liczbę 8bit, gdyż nie dekodujemy adresu). Po załadowaniu portu sprawdzamy, czy odpowiadający przyciskowi bit (tutaj nazwany symbolicznie *INC_BTN*) jest **wyzerowany** - bo to oznacza, że jest **naciśnięty** (zastosowany pullup rezystorowy)

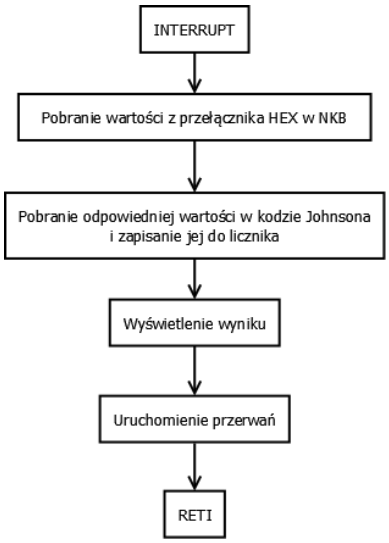
Następnie kontrolujemy stan znacznika, mówiącego o tym, czy czasem nie wyszliśmy dopiero co z przerwania **i dwa przyciski były wciśnięte**. Bit od-

powiadający za ten znacznik to *FLAGA* . Jeśli takie zdarzenie miało miejsce, to omijamy inkrementację.

Ważne jest to, iż przycisk inkrementacji **reaguje zboczem**. Istotna jest zatem pętla *OCZEKIWANIE2*, gdzie musimy odczekać, aż użytkownik puści przycisk.

Po każdym **wciśnięciu i puszczeniu** przycisku inkrementacji **odczekujemy pewien czas**, by minął okres drgania styków. Tego odczekiwania nie stosujemy do przycisku ładowania. On jest asynchroniczny.

Obsługa przerwania maskowalnego



Rysunek 3: Schemat działania procedury przerwania maskowalnego.

Teraz pora na kawałek kodu, czyli co i jak:

```
DS  0x1838-$
;-------------------------------------------------------
; INT, procedura przerwania maskowalnego
;-------------------------------------------------------
INT_LADOWANIE:
  EX  AF, AF
  EXX

  IN  A, (00h) ; pobierz stan HEXa, wartosc w kodzie NKB

  ; tutaj trzeba to przerobic na kod johnsona
  LD  B, 0x07    ; maska trzech pierwszych bitow
  AND B          ; wyzeruj pozostale
  LD  HL, WARTOSCI_JOHNSON ; zaladuj adres poczatku tablicy
  LD  C, A       ; wrzuc nasz kod przesuniecia do C
  LD  B, 0x00    ;  B wyzeruj
  ADD HL, BC     ; dodaj przesuniecie, wynik w HL

  LD  A, (HL)    ; obliczylismy miejsce w tablicy
  LD  (ZMIENNA), A ; teraz ladujemy ta wartosc do zmiennej

  OUT (0x00), A ; wyswietl wynik na ledach

  IN  A, (00h)  ; pobierz stan przycisku
  BIT INC_BTN, A ; sprawdz, czy czasem teraz nie jest wcisly
  JR  NZ, NIEWCISNIETY ; jesli nie, to skocz
  LD  A, FLAGA ; jesli tak, ustaw flage
  LD  (ZMIENNA_INT), A

NIEWCISNIETY:
  EX  AF, AF
  EXX
  EI

  RETI
```

Zgodnie z użytym trybem pracy przerwania maskowalnego `IM 1` wynika, iż procedura jego obsługi musi zostać umieszczona w adresie `0x38` licząc od początku pamięci. W naszym przypadku będzie to `0x1838`, co też wykonuje pierwsza dyrektywa

Po wejściu do przerwania blokowana jest flaga ich dalszej obsługi, przez co nie musimy dodatkowo tego czynić. Jedyne co musimy po wejściu do przerwania uczynić, to przerzucić się na **drugi zestaw rejestrów**, by nie naruszyć struktury działania aktualnie wykonywanego programu. Służą do tego instrukcje `EX, EXX`. Zdecydowaliśmy się wymienić wszystkie rejestry, gdyż będziemy korzystać jeszcze z rejestrów `B, C`.

Samo ładowanie wygląda następująco :

- Zczytaj wartość z przełącznika szesnastkowego

- Zamaskuj tylko trzy pierwsze bity (z 16 mamy tylko 8 kombinacji, więc ignorujemy pozostałe bity) - otrzymujesz numer kombinacji k. Johnsona
- Odczytujesz z tablicy *WARTOSCI_JOHNSON* wartość kodu o danym numerze
- Zapisujesz wartość kodu do zmiennej i na diody

Ztablicowaliśmy kody Johnsona. Jest ich zawsze o wiele mniej niż całkowita ilość kombinacji na n bitach, a sam algorytm generowania kodu o danym numerze jest tylko niepotrzebnym utrudnieniem. Teraz, wystarczy załadować adres początku tablicy do rejestru, dodać do niego przesunięcie i odczytać to, co się znajduje w pamięci.

Zanim wyjdziemy, sprawdzamy, czy czasem **jednocześnie nie jest wciśnięty przycisk inkrementacji**. Jeśli tak, ustawiamy odpowiednią flagę.

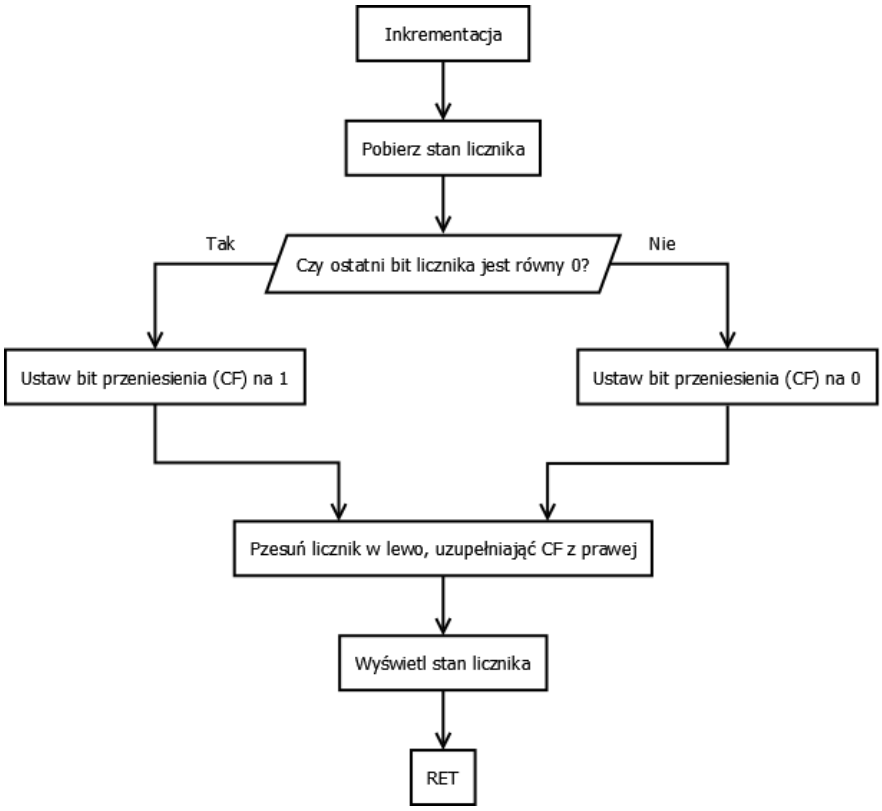
Na koniec, **przywracamy stan rejestrów**, przerzucając się na roboczy zestaw rejestrów. Wychodząc, **aktywujemy przerwania**.

Inkrementowanie

```
INKREMENTUJ:
  PUSH AF
  LD A, (ZMIENNA) ; pobierz licznik
  BIT 3, A ; sprawdz stan czwartego bitu

  SCF
  CCF    ; resetuj carry (1 -> 0)
  ; TUTAJ CF=0
  ; JESLI BIT3, A == 0
  ; TO USTAW CF
  JR NZ, PO_MODYFIKACJI_CF
  SCF    ; ustaw carry

PO_MODYFIKACJI_CF:
  RLA ; przesun w lewo, uzupelnij carry z prawej

  LD (ZMIENNA), A ; uaktualnij zmienna
  OUT (0x00), A ; wyswietl wynik na ledach

  POP AF
  RET
```

Inkrementowanie kodu Johnsona nie sprawia większych trudności. Całość polega na **przesuwaniu w lewo i uzupełnianiu odpowiednim** bitem na początku. Bit ten jest **negacją** tego, który wypada podczas przesuwania. Nasz kod jest 4ro bitowy, więc badamy stan bitu 4 **przed** przesunięciem.

Z80 oferuje instrukcję `RLA`. Powoduje ona przesunięcie w lewo akumulatora z jednoczesnym uzupełnieniem z prawej wartością znacznika *Carry* **przed** wy-



Rysunek 4: Procedura wykonująca inkrementację i wyświetlanie wyniku

konaniem przesuwania. Nowe *Carry* jest ustawiane tym, co wypadło z lewej, ale nas to nie interesuje.

Posiadając wiedzę na temat tego, jaki był bit nr 4 odpowiednio ustawiamy-/resetujemy znacznik *Carry* tak, by instrukcja `RLA` dokonała wtedy pożądanego przez nas uzupełnienia.

Jak na procedurę przystało, korzystamy z akumulatora i rejestru znaczników, które na początku i na końcu funkcji **zachowujemy i pobieramy ze stosu.**

Debouncing

```
;-------------------------------------------------------
; Delay - procedura odczekujaca okolo 50ms
;-------------------------------------------------------
DELAY:
  PUSH  AF ; odloz na stos rejestry
  PUSH  BC

  XOR   A ; wyzeruj akumulator
  LD  B, 0x1E ; w przyblizeniu 50ms

  INCREMENT_OUTER: ; zewnetrzna petla
    INCREMENT_INNER:
      INC   A
      JR  NZ, INCREMENT_INNER
    DEC   B ; dekrementuj B
    JR  NZ, INCREMENT_OUTER ; jesli zero, to koncz

  POP   BC
  POP   AF
  RET
```

Debouncing w naszym przypadku polega na zwykłym odczekaniu w pętli zamierzonego okresu czasu. Na oscyloskopie sprawdziliśmy, iż przy wybranej przez nas wartości uzyskaliśmy opóźnienie w wielkości około 50ms

Samo odczekiwanie to po prostu podwójna pętla w której dekrementowane są rejestry A, B. Jeden obieg pętli zewnętrznej to 256 obiegów pętli wewnętrznej.

Czas 50ms to w przybliżeniu optymalny dla większości prostych przycisków, by te mogły przestać drgać, co potwierdziły też nasze późniejsze testy

Zmienne i stałe użyte w programie

```
;-------------------------------------------------------
; Dane programu
;-------------------------------------------------------
ZMIENNA: ;  licznik
DB 0x00

ZMIENNA_INT: ; znacznik przerwania
DB 0x00

WARTOSCI_JOHNSON: ; stablicowane wartosci kodu Johnsona
DB 0x00, 0x01, 0x03, 0x07, 0x0F, 0x0E, 0x0C, 0x08

INC_BTN:  equ 4 ; stała, numer przycisku
FLAGA:    equ 0 ; stała, numer bitu flagi
```