SDC_One – komputer zdefiniowany programowo z klasycznym mikroprocesorem (6)

Użytkowanie SDC_One

W poprzednich artykułach serii przedstawiliśmy projekt sprzętu i oprogramowania komputera SDC_One. W obecnej części artykułu przedstawimy sposoby programowania komputera przy użyciu łatwo dostępnego oprogramowania działającego na komputerach PC.

SDC_One został zaprojektowany głównie do zastosowań dydaktycznych – praktycznej demonstracji działania mikroprocesora, w tym wykonania instrukcji, trybów adresowania pamięci i obsługi wyjątków. Komputer może być również używany do nauki programowania w językach asemblerowych lub do uruchamiania oprogramowania typowego dla komputerów domowych z lat 1980-tych.

Tworzenie oprogramowania przy użyciu komputera PC

Najprostszym sposobem programowania SDC_One jest użycie do tworzenia programów oprogramowania działającego na komputerze PC.

Listing 1. Plik nagłówkowy dla asemblera TASM zawierający definicje związane ze sterownikiem MINI_IO

```
; SDC_One MiniIO ports and field definitions
; JK & gbm 2016-18
#ifndef MINIO_BASE
#define MINIIO_BASE 0
#endif
;====================================
; MINI IO registers
IO_LEDS = MINIIO_BASE + 0
IO_CON  = MINIIO_BASE + 1
IO_TMR  = MINIIO_BASE + 2
IO_CSR  = MINIIO_BASE + 3
IO_LEDR = MINIIO_BASE + 4
IO_LEDG = MINIIO_BASE + 5
IO_LEDB = MINIIO_BASE + 6
IO_LEDN = MINIIO_BASE + 7
IO_MS_SEC  = MINIIO_BASE + 8
IO_MS_TRK  = MINIIO_BASE + 9
IO_MS_UNIT = MINIIO_BASE + 0ah
IO_MS_CMD  = MINIIO_BASE + 0bh
IO_MS_A0 = MINIIO_BASE + 0ch
IO_MS_A1 = MINIIO_BASE + 0dh
IO_MS_A2 = MINIIO_BASE + 0eh
;====================================
; LED control
#define IO_LEDS_RED (1 << 0)
#define    IO_LEDS_GRN (1 << 1)
#define    IO_LEDS_BLU (1 << 2)
#define    IO_LEDS_NUC (1 << 3)
#define    IO_LEDS_SET (1 << 6)
#define    IO_LEDS_RES (1 << 7)
#define    IO_LEDS_TGL (3 << 6)
; timer period 8ms flag
#define    IO_TMR_X8MS (1 << 7)
; mini_io status register
#define    IO_CSR_RXNE (1 << 0)
#define    IO_CSR_TXE  (1 << 1)
#define    IO_CSR_TI   (1 << 2)
#define    IO_CSR_BTN  (1 << 7)
; Mass storage commands
#define    IO_MS_CMD_RD    1
#define    IO_MS_CMD_WR    2
#define    IO_MS_CMD_SYNC  3
;====================================
```

Do asemblacji potrzeby jest asembler skrośny odpowiedni dla wybranego procesora. Dla procesorów 8-bitowych może to być np. program TASM (Table-driven assembler) – uniwersalny asembler, wyposażony w definicje języków asemblerowych m.in. 8085, Z80 i 6502 lub dowolny inny asembler skrośny działający w użytkowanym przez nas systemie operacyjnym. Istotne jest, by w wyniku translacji można było uzyskać plik wynikowy w jednym z obsługiwanych przez monitor szyny formatów hex – Intel, S-record lub MOS Technology. Jeśli wybrany przez nas asembler generuje tylko pliki binarne, potrzebny będzie dodatkowo program do konwersji na postać .HEX, np. popularny Srecord, dostępny na GitHub.

Wygodnym środowiskiem do tworzenia programów asemblerowych dla MC68k jest dostępny w sieci program Easy68k. Oprócz asemblera z IDE Easy68k zawiera również symulator komputera z procesorem MC68000, który może być używany do celów dydaktycznych.

Asembler TASM

Zaletą programu TASM jest jednolita składnia dyrektyw asemblera, stałych i wyrażeń, niezależna od typu procesora docelowego. Jest więc on szczególnie wygodny, gdy chcemy pisać programy dla kilku wersji SDC_One, wyposażonych w różne procesory. TASM umożliwia posługiwanie się plikami nagłówkowymi, zawierającymi potrzebne w wielu programach definicje, np. adresów portów i masek bitowych. Na **listingu 1** pokazano plik nagłówkowy TASM z definicjami dotyczącymi sterownika MINI_IO, używanego w SDC_One.

Pliki .HEX generowane przez asembler TASM noszą domyślnie niestandardowe rozszerzenie .OBJ. Można zmienić to zachowanie programu, podając jawnie nazwę pliku wyjściowego w linii polecenia.

Pisząc program asemblerowy należy zwrócić uwagę na możliwość uruchomienia go po zainicjowaniu procesora oraz na właściwe dla danego procesora użycie przestrzeni adresowej.

Poniżej przedstawiono zasady tworzenia programów dla różnych procesorów oraz działający na komputerze SDC_One program demonstracyjny napisany w czterech wersjach dla czterech różnych procesorów. Program demonstracyjny wypisuje na terminalu napis, programuje timer na okres 1 s, a następnie co 1 sekundę zmienia kolor świecenia diody RGB i wysyła na terminal kolejną literę z zakresu ‘A’…’P’.

Listing 2. Program demonstracyjny dla mikroprocesora 8085

```
; SDC_One 8085 demo
; gbm 02'2018
; assemble: tasm -85 -x3 sdcdemo_85.asm sdcdemo_85.hex
#include "miniio.h"
    .org    0
start:
    mvi a, IO_TMR_X8MS | 125  ; 125 * 8 ms = 1s
    out IO_TMR    ; setup timer
wait1s:
    in  IO_CSR    ; get timer status
    ani IO_CSR_TI
    jz  wait1s
    out IO_CSR    ; clear timer flag
    in  IO_LEDS   ; get LED state
    adi 'A'       ; convert to letter - increment
    mov b, a      ; save letter in B
    ani 0fh
    out IO_LEDS   ; set LEDs
    in  IO_CSR    ; get console status
    ani IO_CSR_TXE
    jz  wait1s
    mov a, b
    out IO_CON    ; output a letter
    jmp wait1s
    .end
```

Listing 3. Program demonstracyjny dla mikroprocesora Z80CPU

```
; SDC_One Z80CPU demo
; gbm 02'2018
; assemble: tasm -80 sdcdemo_z80.asm sdcdemo_z80.hex
#include "miniio.h"
    .org    0
start:
    ld  a, IO_TMR_X8MS | 125 ; 125 * 8 ms = 1s
    out (IO_TMR), a    ; setup timer
wait1s:
    in  a, (IO_CSR)    ; get timer status
    and IO_CSR_TI
    jr  z, wait1s
    out (IO_CSR), a    ; clear timer flag
    in  a, (IO_LEDS)   ; get LED state
    add a, 'A'         ; convert to letter - increment
    ld  b, a           ; save letter in B
    and 0fh
    out (IO_LEDS), a   ; set LEDs
    in  a, (IO_CSR)    ; get console status
    and IO_CSR_TXE
    jr  z, wait1s
    ld  a, b
    out (IO_CON), a    ; output a letter
    jr  wait1s
    .end
```

Programowanie procesorów 8085 i Z80 CPU

W przypadku procesorów 8085 i Z80 CPU program powinien rozpoczynać się od adresu 0, a dane i stos powinny być umieszczone za programem. Wskaźnik stosu powinien być w programie zainicjowany na adres następny po adresie ostatniej komórki stosu. Jeśli planujemy umieszczenie stosu na końcu przestrzeni adresowej, wskaźnik stosu należy zainicjować na 0.

Listy instrukcji 8085 i Z80 są nadzbiorami listy instrukcji 8080. Każdy program używający wyłącznie instrukcji 8080 może być zatem uruchomiony na obu tych mikroprocesorach. Przy programowaniu Z80 używa się jednak na ogół języka asemblerowego Z80, znacznie bardziej przejrzystego od asemblera 8080/8085. Również oprogramowanie SDC_One dla Z80 zawiera deasembler języka asemblerowego Z80.

Listing 2 przedstawia program demonstracyjny zapisany w asemblerze 8085, a **Listing 3** – ten sam program w reprezentacji asemblerowej Z80.

Programowanie 6502

Dla 6502 pod adresem 0xFFFC należy umieścić adres początku programu. Ponieważ pierwsze 256 bajtów przestrzeni adresowej służy do przechowywania najczęściej używanych danych, a kolejne 256 bajtów zajmuje stos, typowo program

Listing 4. Program demonstracyjny dla mikroprocesora W65C02S

```
; SDC_One 65c02 demo
; gbm 02'2018
; assemble: tasm -65 -x3 sdcdemo_65.asm sdcdemo_65.hex
#define MINIIO_BASE    0fe00h
#include "miniio.h"
    .org    0ff00h
start:
    lda #IO_TMR_X8MS | 125  ; 125 * 8 ms = 1s
    sta IO_TMR  ; setup timer
wait1s:
    lda IO_CSR  ; get timer status
    and #IO_CSR_TI
    beq wait1s
    sta IO_CSR  ; clear timer flag
    lda IO_LEDS ; get LED state
    clc
    adc #'A'    ; convert to letter - increment
    tax         ; save letter in X
    and #$0f
    sta IO_LEDS     ; set LEDs
    lda IO_CSR      ; get console status
    and #IO_CSR_TXE
    beq wait1s
    stx IO_CON  ; output a letter
    bra wait1s
; reset vector
    .org    0fffch
    .dw start
    .end
```

powinien się zaczynać od adresu nie mniejszego niż 0x200. Krótkie programy mogą być zawarte w ostatniej 256-bajtowej stronie pamięci. Strona przedostatnia, rozpoczynająca się od adresu 0xFE00, jest zajęta przez sterownik wejścia-wyjścia MINI_IO. Wskaźnik stosu nie musi być inicjowany, ponieważ stos tworzy bufor cykliczny zajmujący całą drugą stronę pamięci i w praktyce jedynym sposobem korzystania ze danych na stosie są operacje PUSH i POP. Program demonstracyjny w wersji dla 6502 przedstawiono na **listingu 4**. W porównaniu z Z80 uwagę zwraca bardzo prosta lista instrukcji i prosty, chociaż niezbyt intuicyjny sposób ich zapisu w języku asemblerowym 6502.

Programowanie MC68008

Program demonstracyjny dla MC68008 został napisany w środowisku Easy68K (**listing 5**). W MC68008 pod adresem 0 należy umieścić początkową wartość wskaźnika stosu systemowego, a pod adresem 4 – początkową wartość licznika instrukcji. Program powinien zaczynać się za ostatnim używanym elementem tablicy adresów procedur obsługi wyjątków.

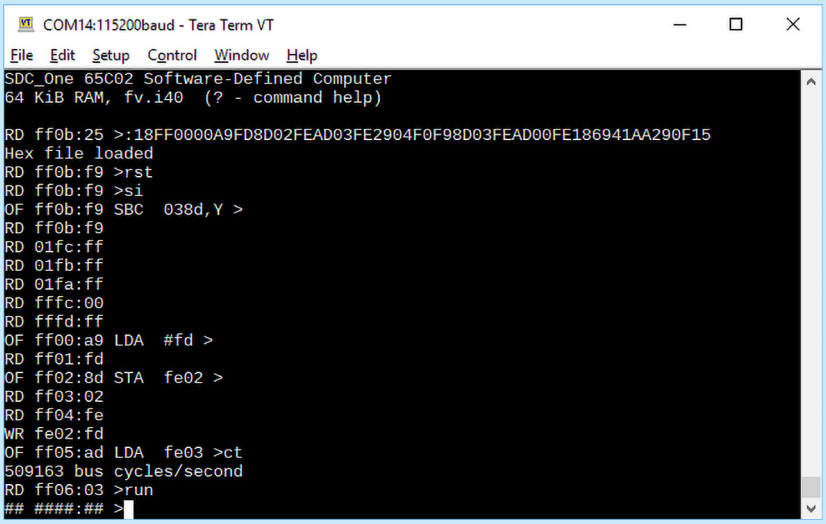
Uzyskany w wyniku translacji plik w formacie S-record ładuje się do SDC_One tak samo, jak pliki Intel .HEX – przez przeciągnięcie go do okna terminala monitora sprzętu.

Listing 5. Program demonstracyjny dla mikroprocesora MC68008

```
; SDC_One MC68008 demo
; gbm 02'2018
; assemble with Easy68K
;=======================================
MINIIO_BASE equ -256
; MINI IO registers
IO_LEDS     equ MINIIO_BASE+0
IO_CON      equ MINIIO_BASE+1
IO_TMR      equ MINIIO_BASE+2
IO_CSR      equ MINIIO_BASE+3
IO_TMR_X8MS equ 1<<7
IO_CSR_RXNE equ 1<<0
IO_CSR_TXE  equ 1<<1
IO_CSR_TI   equ 1<<2
;=======================================
    Org     0
    dc.l    $1000  ; initial SSP value
    dc.l    start  ; program entry point
    org     $400
start:
    move.b  #(IO_TMR_X8MS | 125), IO_TMR          ; 125 * 8 ms = 1s
wait1s:
    btst.b  #2, IO_CSR  ; get timer status
    beq.b   wait1s
    move.b  #IO_CSR_TI, IO_CSR
    move.b  IO_LEDS, d0 ; get LED state
    add.b   #'A', d0    ; convert to letter - increment
    move.b  d0, d1      ; save letter
    andi.b  #$0f, d0
    move.b  d0, IO_LEDS ; set LEDs
    btst.b  #1, IO_CSR  ; get console status
    beq.b   wait1s
    move.b  d1, IO_CON  ; output
    bra.b   wait1s
    end
```

Uruchamianie i monitorowanie programów

Generowany w wyniku asemblacji programu plik ładowalny .HEX jest plikiem tekstowym zawierającym reprezentację zawartości pamięci zapisaną w postaci liczb szesnastkowych. W celu załadowania programu zapisanego w pliku .HEX do pamięci komputera należy przesłać go przez terminal monitora systemu. Monitor reaguje na rekordy plików .HEX jako na polecenia ładowania pamięci zawartością rekordów. Jeżeli używamy terminala TeraTerm lub podobnego wyposażonego w funkcję drag-and-drop, w celu przesłania pliku wystarczy uchwycić go kursorem myszy i upuścić w oknie terminala. Ze względu na zależności czasowe w monitorze systemu zalecane jest ustawienie w programie terminala niezerowego odstępu czasowego po transmisji końca wiersza.



Rysunek 1. SDC_One z W65C02S – ładowanie programu, start procesora i test wydajności

Po załadowaniu programu do pamięci możemy go uruchomić, resetując procesor poleceniem monitora rst. Po wykonaniu polecenia monitor zatrzyma procesor w pierwszym rozpoczętym cyklu transmisji. W zależności od typu procesora będzie to zwykle cykl pobrania kodu operacyjnego pierwszej instrukcji lub cykl pobrania adresu, który ma być załadowany do rejestru SP lub PC procesora. Procesor 6502 po zainicjowaniu wykonuje jedną instrukcję i kilka pustych cykli odczytu z obszaru stosu, a następnie przechodzi do właściwego pobrania adresu początkowego programu (**rysunek 1**).

Monitor sprzętu umożliwia m.in. pracę krokową programów w cyklach szyny (polecenie sc) lub w cyklach instrukcyjnych (polecenie si). Oba polecenia są powtarzane samoczynnie po wciśnięciu klawisza Enter, co ułatwia prezentację działania procesora. Polecenie run powoduje uruchomienie programu w trybie pracy ciągłej – z maksymalną wydajnością komputera. Mikroprocesor MC68008 jest wyposażony w mechanizm wczesnego pobierania instrukcji, którego działanie łatwo można zaobserwować przy użyciu monitora sprzętu. Procesor zawsze pobiera pierwsze słowo następnej instrukcji przed wykonaniem odwołań do danych wynikających z bieżącej instrukcji.

Grzegorz Mazur