Bellinzona SAM

Z80µPC Single Board Computer Development

Naoki Pross

7 settembre 2017

Sommario

Lo Zilog Z80 è un processore a 8 bit che fu introdotto nel 1976 che ebbe un grandissimo successo nel mondo dell'elettronica e dell'informatica nella fine del 20esimo secolo. In memoria di questo pioniere dell'industria dei sistemi informatici questo progetto documenta la realizzazione di un microcomputer a scopo generico a base di esso. L'obiettivo primario dunque è di realizzare una scheda simile ad una motherboard dei computers venduti all'epoca completa di RAM, ROMs, interfacce seriali e altri circuiti di supporto. Successivamente per l'aspetto software il progetto deve implementare i drivers per ogni circuito presente sulla scheda in modo da semplificare la programmazione. L'obiettivo opzionale del progetto, una volta terminata la costruzione hardware, è di realizzare una kernel monolitica che offre funzioni minimali simili ad un sistema UNIX, quali processi, filesystem, memory management e drivers.

1 Specifiche tecniche dello Z80

Lo Z80 è un processore molto minimalistico se paragonato a ciò che si trova oggi sul mercato dei microcontrollori. Per il progetto Z80µPC la CPU in uso è il modello originale `Zilog Z8400` che non dispone di sistemi integrati come i modelli SoC odierni. Le specifiche più importanti sono elencate a seguire.

- Architettura a 8 bit con bus a 16 bit, 64K indirizzi indirizzabili
- Registri a 16 bit per `SP`, `PC` e registri di utilizzo generico a 8 bit `A..F` combinabili a coppie `AF`, `BC`, `..` per utilizzare valori a 16 bit
- Clock fino a 8 MHz
- Segnali di controllo tra cui $\overline{\texttt{RD}}$, $\overline{\texttt{WR}}$, $\overline{\texttt{IOREQ}}$, $\overline{\texttt{MREQ}}$ e $\overline{\texttt{RST}}$
- Interrupts mascherabili e non con vettore a 8 bit

2 Architettura di base

Il minimo necessario per far funzionare un computer con lo Z80 sono una ROM e una RAM, ma per il mio progetto ho scelto di aggiungere dell'hardware aggiuntivo per lo sviluppo si sistemi più complessi per apprendere conoscenze sia nel mondo dell'elettronica che dell'informatica. Per questa ragione lo Z80µPC possiede i seguienti componenti:

ROM	**M28C64**	EEPROM da 8KB x 8 bit (64K) per il BIOS / Bootloader / OS installata doppia per avere 16KB
RAM	**HM62256B**	SRAM da 32KB x 8bit (256K)
CTC	**Z8430**	Counter timer circuit ufficiale di Zilog a 4 canali che permette di essere programmato
PIO	**Z8420**	Parallel input/output controller di Zilog per avere un intefaccia digitale con due porte da 8 bit
MMU	**M4-32/32-15JC**	CPLD programmabile che implementa una memory management unit semplificata in grado di gestire i 5 bit più significativi della linea di indirizzi
USART	**TL16C550C**	Interfaccia USART per poter comunicare utilizzando il protocollo RS232

Oltre a tutto ciò per uno scopo formativo lo Z80µPC dispone anche di strumenti da debug e analisi per comprendere ogni operazione del processore. Il modello di Z80 scelto è in grado di utilizzare un clock fino a 8MHz, ma non definisce un minimo dunque sono presenti 3 circuiti che generano 3 clock di velocità differenti.

0Hz Questo clock è un bottone che permette di creare manualmente le pulsazioni del clock, per poter analizzare ogni istruzione

200Hz Mediante un classico circuito con un LM555 si ha un clock da 200Hz per eseguire i programmi a velocità rallentata

4MHz Clock per esecuzione a velocità piena (normale)

Una seconda disposizione per aiutare la comprensione del funzionamento del processore è data da 6 display a 7 segmenti che durante l'esecuzione rallentata o a step (bottone) visualizzano i bytes presenti sulla bus di indirizzi a 16 bit e sul bus di dati a 8 bit.

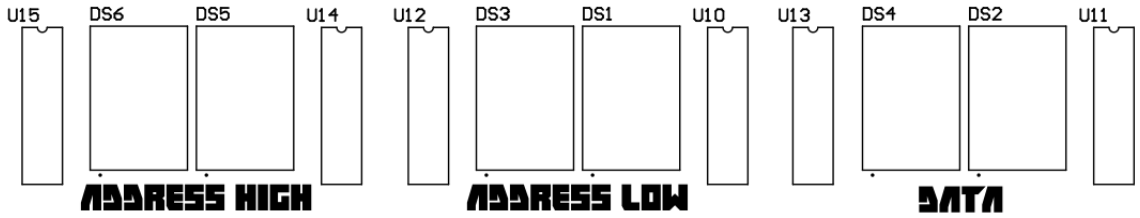


Figura 1: Display a 7 segmenti per visualizzare il flusso di dati della CPU

3 Memory Management Unit

Alcuni modelli successori dello Z8400 implementavano una MMU SoC che permetteva di indirizzare un address space di dimensione maggiore. Per lo Z80µPC non necessito di un indirizzamento più grande ma piuttosto sono interessato dalle operazioni di gestione della memoria di una MMU simile a ciò che accade nelle architetture X86. Nelle architetture odierne basate sull'X86/64 è presente un sistema di traslazione di indirizzi di memoria da virtuale a fisica. Con lo scopo di trarne solamente i vantaggi più fondamentali lo Z80µPC implementa nella CPLD MMU un sistema basilare di gestione di pagine di memoria con traslazione di indirizzi in modo da poter allocare più programmi nella RAM anche se il sistema non implementa il multitasking.

3.1 Address Space

La funzione primaria della MMU è di mappare i dispositivi I/O e le memorie nell'address space. Nell'implementazione reale la MMU controlla i segnali $\overline{CS}$ seguendo una logica combinatoria molto semplice che controlla se l'indirizzo sul bus si trova in una zona definita per un dispositivo. L'address space si presenta dunque nella seguente maniera, per cui la ROM occupa il primo quarto, i dispositivi mappabili il secondo quarto e la RAM la metà restante. Essendo un progetto pensato per essere esteso 16KB sono liberi per mappare dispositivi esterni collegati attraverso il connettore DIN41612.

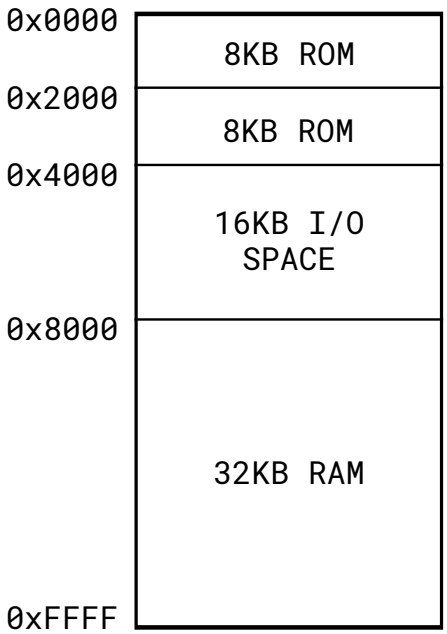
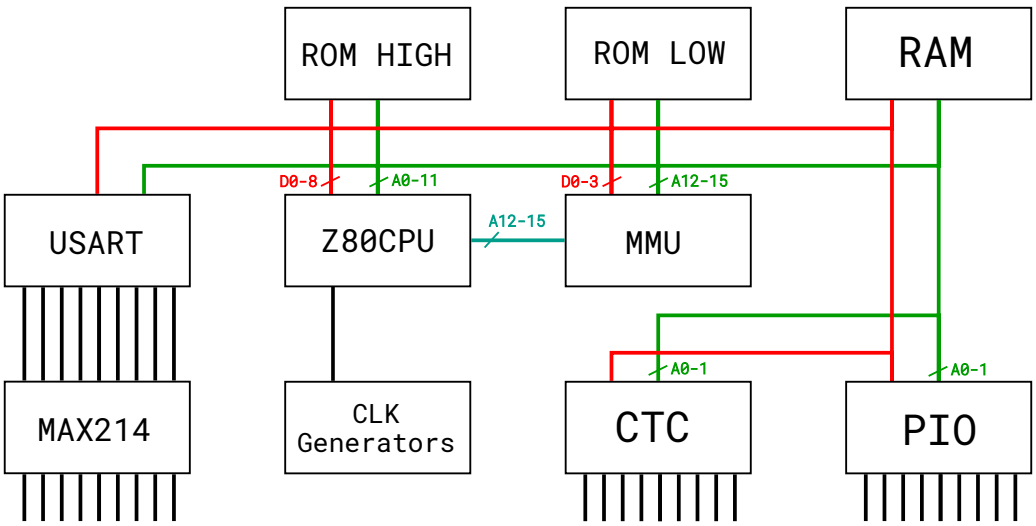


Figura 2: Address space dello Z80µPC

3.2 Page Table

Per poter controllare la traslazione degli indirizzi la MMU dispone di una Page Table a cui è possibile accedere attraversso un certo indirizzo nell'I/O space. La page table di 5 bit permette la gestione delle regioni di memoria da impostare per dei determinati processi. Questa funzione è importante perchè permette la separazione dello stack e della memoria della kernel dai programmi normali. Per lo Z80µPC potrebbe sembrare eccessivo ma essendo uno strumento per apprendere le fondamenta dell'elettronica e dell'informatica è interessante implementare questa funzionalità che comunque se necessario può essere disabilitata.

4 Schema a blocchi



5 Software / Sistema operativo

Negli sviluppi più recenti intorno allo Z80 esso veniva utilizzato come un microcontrollore anzichè come processore da computer, per questa ragione non sono presenti molti sistemi operativi per questa piattaforma. Dunque per lo Z80µPC il progetti implementa un sistema operativo soprannominato `HelvetiOS` con le funzioni minime necessarie come un interfaccia seriale a comandi e un meccanismo per caricare i programmi.

5.1 Componenti di base

Per garantire un funzionamento minimo il sistema `HelvetiOS` deve offrire drivers e utility di base quali:

- USART driver and API
- PIO driver and API
- CTC driver and API
- Bootloader
- Program launcher
- Shell-like interface

5.2 Interfacce dell'API

Nel corso dello sviluppo questa sezione verrà continuamente espansa per documentare le interfacce dei vari drivers.

5.2.1 USART

```
void usart_set_baudrate(uint16_t baudrate);
void usart_set_parity(int mode);
void usart_set_stop_bits(int count);
void usart_set_word_length(int length);
void usart_set_autoflow(int mode);

inline void usart_init(uint16_t baudrate, int parity, int stop_bits);

void usart_transmit(uint8_t data);
uint8_t usart_receive();

int usart_write(uint8_t *data, size_t size);
int usart_read(uint8_t *buffer, size_t count);
```

5.3 Toolchain per la compilazione

Per compilare il software del progetto si utilizza SDCC, un progetto open-source che supporta la compilazione di binari per l'architettura dello Z80. Nella mia configurazione utilizzo GNU make con il seguente makefile.

```
1  ####
2  # Source code settings
3  #
4  OSNAME := helvetiOS
5
6  CSOURCES    := $(wildcard kernel/*.c) $(wildcard libc/*.c)
7  OBJECTS     := $(patsubst %.c,build/%.rel,$(CSOURCES))
8  HEXFILE     := build/$(OSNAME).hex
9  BINARY      := build/$(OSNAME).bin
10
11 ###
12 # Compiler settings
13
14 CC      := sdcc
15
16 CFLAGS  := -mz80 \
17             -I kernel/include -I libc/include -DDEBUG
18
19 LDFLAGS := -mz80 --no-std-crt0 crt0.rel \
20             --code-loc 0x0800 --data-loc 0x8000
21
22 .PHONY: dirs dis clean
23 all: $(BINARY)
24
25 # build binary
26 $(BINARY): $(OBJECTS) dirs
27     $(CC) $(LDFLAGS) $(OBJECTS) -o $(HEXFILE)
28     xxd -r -p $(HEXFILE) $(BINARY)
29
30 $(OBJECTS): build/%.rel : %.c $(CSOURCES) dirs crt0.rel
31     $(CC) $(CFLAGS) -c $< -o $@
32
33 crt0.rel: crt0.s
34     sdasz80 -o $<
35
36 dirs:
37     mkdir -p build build/kernel build/libc
38
39 dis: $(BINARY)
40     z80dasm -a -g 0h $< -o $(OSNAME).s
41
42 clean:
43     - rm -rd build/*
44     - rm $(OSNAME).s
45     - rm crt0.rel
```