

Z80 μ PC Single Board Computer Development

Naoki Pross

14 novembre 2017

Sommario

Lo Zilog Z80 è un processore a 8 bit che fu introdotto nel 1976 ed ebbe un grandissimo successo nel mondo dell'elettronica e dell'informatica dagli anni 70 a 90. 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.

Indice

| | |
|--|----------|
| 1 Hardware | 3 |
| 1.1 Specifiche tecniche dello Z80 | 3 |
| 1.2 Componenti e modello di design | 3 |
| 1.3 Schema a blocchi | 4 |
| 1.3.1 Implementazione dei generatori di clocks | 4 |
| 1.3.2 Uscite verso l'esterno (DIN 41612) | 4 |
| 1.4 Address space | 5 |
| 1.5 Memory management unit | 5 |
| 2 Software | 8 |
| 2.1 Organizzazione del codice sorgente C | 8 |
| 2.2 C Toolchain | 8 |
| 2.3 CRT0 per lo Z80 | 9 |
| 2.4 Codice sorgente VHDL | 10 |
| 2.4.1 Programmazione della MMU | 10 |

1 Hardware

1.1 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 Z80 μ PC la CPU in uso è il modello originale Zilog Z8400 che non dispone di moduli aggiuntivi integrati come i modelli SoC odierni. La scelta di una CPU tanto semplice è la conseguenza del design didattico del progetto, inoltre senza alcun dispositivo interno lo Z8400 si presenta con un address space completamente vuoto, ad eccezione del punto d'inizio e i vettori di reset.

Lo Z80 utilizza I/O paralleli sia per la linea a 16 degli indirizzi che per la linea dati a 8 bit e dispone di 6 registri 8 bit ad utilizzo generico combinabili in coppie per ottenere un valore a 16 bit. Per il controllo dei dispositivi esterni, come lettura e scrittura esso possiede delle linee di controllo dedicate come \overline{RD} , \overline{WR} , \overline{MREQ} , ecc. In quanto instruction set, lo Z80 ha 158 istruzioni possibili di cui 78 sono un sottoinsieme dello 8080A, architettate per poter mantenere una retrocompatibilità.

Tabella 1: Riassunto delle specifiche

| | |
|-------------------------|---------------------------|
| Dimensione Indirizzi | 16 bit |
| Dimensione Dati (word) | 8 bit |
| Spazio Indirizzabile | 64 KB |
| Registri Generici 8 bit | 6 (A..F) |
| Registri 16 bit | 2 (SP, PC) |
| Clock speed | 8 MHz, 6MHz, 4MHz, 2.5MHz |

1.2 Componenti e modello di design

Il minimo necessario per far funzionare uno Z80 sono una RAM ed una ROM, ma avendo a disposizione altri dispositivi I/O lo Z80 μ PC dispone anche di una porta seriale, di una porta parallela e di un counter timer; Hardware che si presenta normalmente all'interno di microcontrollori odierni.

Il design dello Z80 μ PC è costruito sulla falsa riga di un Arduino o di un EasyPIC con l'aggiunta di funzionalità a scopo didattico quali; la possibilità di cambiare la velocità di clock tra 4MHz, 200Hz o manuale (mediante un bottone sulla scheda) e una serie di display a 7 segmenti per vedere in tempo reale i valori sui bus degli indirizzi e dei dati.

- 0Hz** Il clock manuale è un bottone che permette di creare le pulsazioni, per poter analizzare ogni istruzione
- 200Hz** Mediante un classico circuito con un LM555 si ha un clock per eseguire i programmi a velocità rallentata
- 4MHz** Clock per esecuzione a velocità piena (normale)

Tabella 2: Lista dei 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 programmabili |
| PIO | Z8420 | Parallel input/output controller di Zilog per avere un interfaccia 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 |

1.3 Schema a blocchi

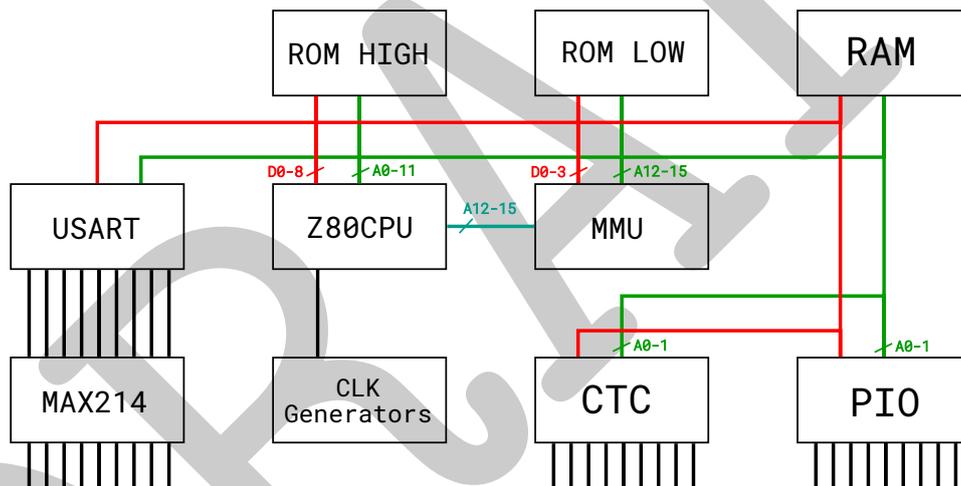


Figura 1: Schema a blocchi del Z80μPC

1.3.1 Implementazione dei generatori di clocks

1.3.2 Uscite verso l'esterno (DIN 41612)

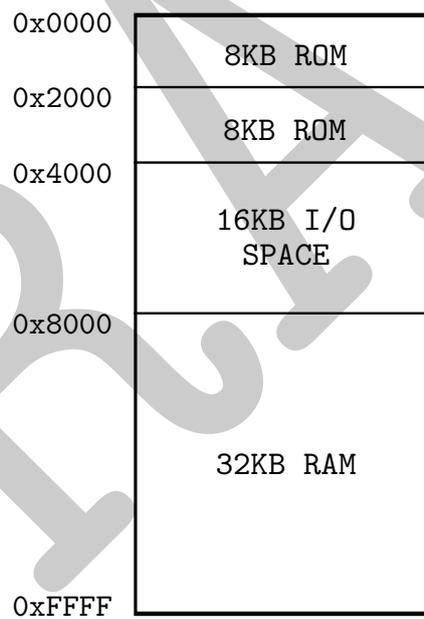
Per poter estendere lo Z80μPC a lato è presente un connettore DIN 41612 per poter interfacciare schede di estensione della funzionalità siccome il progetto lascia liberi la maggior parte dei 16KB dell'*i/o space*, ovvero la parte di address space in cui è previsto di mappare dispositivi esterni.

Tabella 3: Mappatura dei segnali nel connettore DIN 41612

| | | | | | | | | | | | | | | | |
|-------------------|-------------------|-----|-----------------|---------------------|-----|---------------------|-----------------|-----|-----|------------------|------------------|-------------------|------------------|-------------------|-------------------|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| GND | GND | VCC | VCC | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | A0 | GND | A2 | A1 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| A4 | A3 | A6 | A5 | A8 | A7 | A10 | A9 | 12 | A11 | A14 | A13 | | A15 | | |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
| \overline{RD} | | | \overline{WR} | | GND | | $\overline{M1}$ | | GND | \overline{INT} | \overline{RST} | \overline{MREQ} | \overline{NMI} | \overline{HALT} | \overline{TORQ} |
| 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 |
| \overline{RFSH} | \overline{WAIT} | GND | | \overline{BUSREQ} | | \overline{BUSACK} | | CLK | | | | VCC | VCC | GND | GND |

1.4 Address space

L'address space dello z80 è in grado di indirizzare 64 KB; la metà di questi è assegnata alla RAM e un quarto è della ROM. La ROM è minore della RAM perché è intesa per essere un bootloader piuttosto che un vero e proprio OS.

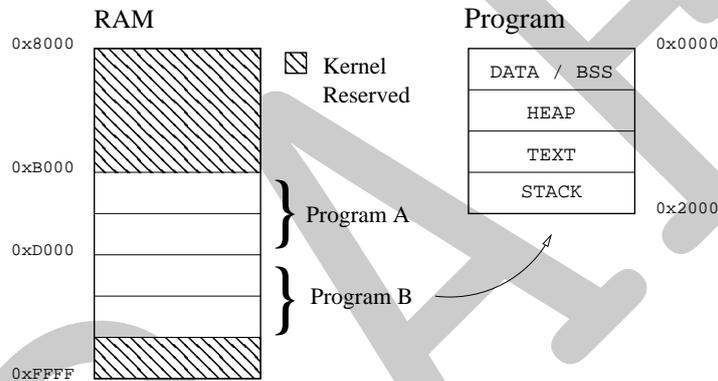


1.5 Memory management unit

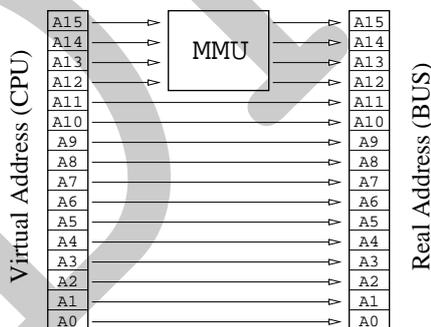
Alcuni modelli successivi dello Z8400 implementavano una MMU (Memory Management Unit) SoC che permetteva di ampliare la dimensione dell'address space, permettendo quindi di mappare più memorie o dispositivi separati negli stessi indirizzi. Lo Z80 μ PC però cerca di imitare un'architettura più simile ad un computer X86 in cui la MMU viene utilizzata per la gestione delle *pagine* di memoria. Il concetto di pagine (pages in inglese) è

necessario per sistemi con un supporto per il multitasking. La conseguenza di questo design implica però che sia introdotto anche il concetto di *virtual address space* siccome, potendo reindirizzare a piacimento gli indirizzi, diventa possibile e conveniente offrire ai programmi eseguiti la stessa struttura di memoria indipendentemente dalla loro vera locazione in memoria.

La struttura designata per il progetto è la seguente. È resa possibile solamente l'esecuzione di 2 programmi semi-paralleli, perchè l'implementazione di uno scheduler è troppo complessa con il CTC a disposizione. Si può quindi avviare due programmi ma solamente uno è in esecuzione ed utilizza il 100% del tempo della CPU. Purtroppo le limitazioni di questo design non permettono di offrire funzionalità importanti della kernel, ma in un futuro si potrebbe risolvere il problema estendendo la piattaforma. Infatti grandi spazi dell'iospace sono stati lasciati liberi per l'aggiunta di dispositivi esterni.



I due programmi in 'esecuzione' sono salvati nella RAM alle locazioni 0xB000 e 0xD000 con 8 KB di RAM per uno. Per passare da un programma all'altro la kernel offre delle system calls che modificano la *page table* all'interno della MMU in maniera da mappare l'indirizzo B000 o D000 a 0000.



Example program running at address 0xB000

Virtual address
LD (#0x1200), #0xFE

Real Instruction
LD (#0xC200), #0xFE

Naturalmente ciò significa che il programma non ha l'accesso all'iospace, perciò per le operazioni con i dispositivi è necessario implementare un driver nella kernel. Nel caso in cui si volesse creare dei drivers userspace la kernel

dovrà mettere a disposizione delle system calls che permettano di modificare la page table in maniera indiretta.

DRAFT

2 Software

2.1 Organizzazione del codice sorgente C

Il codice sorgente dell'intero progetto è contenuto nella cartella `sw/z80`.

- `arch` – Contiene headers e codice essenziale che descrive la configurazione del dispositivo (es: gli indirizzi dei dispositivi).
- `drivers` – Contiene il codice per i drivers dei vari dispositivi, il tutto viene compilato in una libreria statica.
- `kernel` – Contiene il codice della kernel monolitica del progetto.
- `libc` – Contiene delle implementazioni parziali di alcune funzioni della standard library. Non più utilizzata.
- `tests` – Contiene delle test units per controllare individualmente parti del progetto.

2.2 C Toolchain

Per compilare il codice per lo Z80 μ PC è necessario utilizzare un *cross compiler* che sia in grado di compilare per l'architettura dello z80. Per questo progetto si è scelto di utilizzare SDCC (Small Device C Compiler), siccome è un progetto ancora in sviluppo attivo ed è utilizzato anche per compilare in molte altre piattaforme. Per compilare un codice sorgente in un object con SDCC per lo Z80 si utilizza il seguente comando:

```
$ sdcc -mz80 -pedantic --no-std-crt0 <crt0> \  
    --allow-unsafe-read \  
    -I. -c <source_file> -o <object_file>
```

In cui `<source_file>` è il documento con il codice sorgente e `<object_file>` è il nome dell'object (solitamente lo stesso del sorgente con l'estensione `.o`). La flag `--allow-unsafe-read` è permessa solamente nella compilazione per Z80 e serve ad indicare al compiler che le letture da locazioni arbitrarie di memoria sono ammesse (normalmente non lo sono). Per linkare gli objects e generare un eseguibile si utilizza

```
$ sdcc -mz80 --no-std-crt0 <crt0> \  
    --code-loc=<addr> <objects> -o <hexfile>
```

```
$ makebin -s <rom_size> -yo 1 -ya 1 <hexfile> <binary>
```

Le flags `-yo 1 -ya 1` specificano rispettivamente il numero di banchi di ROM e di RAM.

2.3 CRT0 per lo Z80

In C il CRT0 è un insieme di routines che vengono eseguite prima del codice C che servono ad inizializzare il sistema. Nel caso dello Z80 μ PC è utilizzato per inizializzare lo stack pointer e per indicare al linker come organizzare i settori dell'eseguibile. Un esempio di crt0 utilizzato per il progetto:

```
1 | .module crt0
2 | .area _HEADER (ABS)
3 |
4 | ;; Reset vectors
5 | .org 0
6 | jp init
7 |
8 | ; the instruction 0xff (not written)
9 | ; resets to this location
10 | .org 0x38
11 | jp init
12 |
13 | ;; main code
14 | .org 0x100
15 | .globl _main
16 |
17 | init:
18 | ;; Set stack pointer directly above top of memory.
19 | ld sp,#0xffff
20 |
21 | ;; Start of the program
22 | call _main
23 | jp _exit
24 |
25 | _exit:
26 | halt
27 | jp _exit
28 |
29 | ;; Ordering of segments for the linker.
30 | .area _HOME
31 | .area _CODE
32 | .area _INITIALIZER
33 | .area _GSINIT
34 | .area _GSFINAL
35 |
36 | .area _DATA
37 | .area _INITIALIZED
38 | .area _BSEG
39 | .area _BSS
40 | .area _HEAP
```

Il CRT0 essendo scritto in assembly deve essere compilato prima utilizzando un assembler per lo Z80, per esempio quello fornito in SDCC.

```
$ sdasz80 -o <crt0.s>
```

Quindi l'argomento `--no-std-crt0 <crt0>` per il compiler descritto precedente non è assolutamente necessario ma è consigliato siccome permette di avere un controllo maggiore del contenuto dell'eseguibile.

2.4 Codice sorgente VHDL

Il codice sorgente in VHDL per la CPLD utilizzata come address decoder e MMU, è contenuto nella cartella `sw/cpld`. La toolchain utilizzata è quella offerta da Lattice.

2.4.1 Programmazione della MMU

Glossario Tecnico

Address Space

In informatica l'*address space* è un intervallo di indirizzi che possono corrispondere a indirizzi in rete, regioni di un dispositivo, di una memoria o di un qualsiasi altro dispositivo fisico o logico. Per questo progetto *address space* si riferisce all'intervallo indirizzabile dal processore, ovvero 2^{16} locazioni siccome il sistema dispone di un bus a 16 bit.

Virtual Address Space

Il virtual address space è un livello di astrazione che estende l'address space utilizzato nelle architetture moderne (X86 / 64). In un sistema senza VAS, i programmi caricati in memoria 'vedono' l'indirizzo in cui si trovano e devono perciò essere caricati in una regione di memoria continua. In un sistema con VAS invece al programma figura sempre di avere l'intero address space disponibile per se, e la MMU si occupa di traslare gli indirizzi virtuali in indirizzi reali prima che essi raggiungano l'hardware.

Registro

Un registro è un dispositivo di memoria in cui è possibile può leggere e/o scrivere un certo valore. Normalmente in un computer / microcontrollore la dimensione della memoria è data dall'architettura, dunque 8, 16, 32 o 64 bits. In questo documento viene viene comunemente utilizzato per riferirsi ad una memoria di un dispositivo fisico come la CPU o un IC seriale.

Toolchain

Da Wikipedia [1]: In ambito software, una toolchain l'insieme dei programmi (tools) usati nello sviluppo di un prodotto (tipicamente un altro programma o sistema di programmi). I tool possono essere utilizzati in catena, in modo tale che l'output di ciascun tool rappresenti l'input per il successivo, ma il termine utilizzato in maniera pi estesa per riferirsi, pi in generale, a qualunque insieme di tool di sviluppo collegati tra loro.

Stack Pointer, Program Counter

Lo Stack Pointer è un registro a 16 bit in cui è salvato l'indirizzo della fine (più in alto, ultimo inserito) dello stack. Il program counter, un altro registro a 16 bit contiene l'indirizzo da cui leggere della prossima istruzione da eseguire.

Riferimenti bibliografici

[1] <https://it.wikipedia.org/wiki/Toolchain>

[2] **Paolo Di Leo**, Z80 Microcomputer didattico, Gennaio 2016, Libri SANDIT, ISBN 978-88-6928-142-6

DRAFT