*DISPENSA (relativa a strumenti di laboratorio connessi ad argomenti svolti in "Sistemi e Reti")*

# I SISTEMI DI SVILUPPO

In ambito informatico un sistema di sviluppo è un complesso costituito da software ed hardware in grado di mettere un programmatore (progettista) nelle condizioni di creare nuovo software (o sistemi HW/SW) facendo uso di un certo linguaggio di programmazione.
Se il **linguaggio in uso è ad alto livello**, il sistema di sviluppo sarà costituito essenzialmente dal software di supporto (Turbo Pascal, Turbo C, Lazarus, Delphi e compilatori in genere) e dall'hardware a cui è destinato (Pc, home computer ecc.). Diversamente (linguaggi assembly), sarà necessario oltre al software di supporto (editor, assemblatore, linker, loader) la disponibilità dello specifico sistema hardware su cui si dovrà successivamente impiantare il programma. E' infatti nella logica dell'assembler il contatto diretto con la macchina, strumento indispensabile per le operazioni di debug (ovvero di messa a punto del prodotto).

I sistemi di **sviluppo Z80** qui presi in esame sono tre: il Nanocomputer Z80, il simulatore software CPM/Z80 e l'emulatore Z80 Deneb. Tali sistemi di sviluppo hanno caratteristiche diverse e saranno qui esaminati brevemente, una descrizione più dettagliata sarà eventualmente disponibile più avanti su apposite dispense.

Il **Nanocomputer**, strumento dalle applicazioni puramente didattiche, è un elaboratore completo basato su $\mu$p Z80. La scomodità del sistema deriva principalmente dal fatto che:
1) Non è dotato di programmi applicativi tranne un rudimentale S.O. (detto "monitor") che gestisce le risorse hardware di I/O disponibili (tastierino HEX, 4 display 7-segmenti e l'interfaccia seriale).
2) Ogni applicazione HW/SW che si realizza necessita di un proprio Nanocomputer completamente dedicato, con conseguente decollo dei costi se si intende operare a livello professionale e non didattico.
3) Il nanocomputer può essere programmato solo in linguaggio macchina. Questo comporta che il più delle volte si debba creare ed assemblare manualmente il programma, per poi inserirlo da tastiera in formato esadecimale. L'operazione può essere evitata realizzando ed assemblando il programma su PC, e trasferendolo con un opportuno interfacciamento sul nanocomputer.

Il **simulatore CPM/Z80** è un programma per PC che è in grado di emulare il funzionamento di un vecchio (circa 1980) Personal Computer con sistema operativo CPM. Visto che tali macchine erano basate su architettura Z80 e che per emulare tale ambiente è necessario provvedere a simulare l'esecuzione di programmi eseguibili su di questo microprocessore (ovvero in linguaggio macchina Z80), il pacchetto dispone della possibilità di simulare completamente a livello software il funzionamento di una CPU Z80 (a parte chiaramente le istruzioni di I/O che necessitano per essere eseguite dell'effettiva presenza dell'hardware esterno). E` quindi per noi possibile utilizzare tale pacchetto come simulatore software di programmi linguaggio macchina Z80, precedentemente tradotti dal linguaggio assembly da un cross assembler (lo stesso usato per il sistema emulatore Deneb). Tale sistema quindi non permette lo sviluppo completo di applicazioni Hw/Sw.

L'**emulatore** è un potente strumento professionale di sviluppo in quanto:
1) È a diretto contatto con il PC. Questo comporta la possibilità di gestire il sistema avendo a disposizione ogni applicativo disponibile su PC per editare, assemblare, linkare, caricare il programma ideato, e di effettuare tutte la operazioni di debug con le facilitazioni garantite dal PC.
2) E' uno strumento universale. Con un solo emulatore si possono progettare e rendere funzionanti un numero infinito di sistemi indipendenti. Infatti una volta sviluppato un progetto Hw/Sw è possibile scindere l'hardware esterno realizzato dall'I.C.E. (In Circuit Emulator) dell'emulatore, sostituendolo con un vero microprocessore, ottenendo così una scheda indipendente di costo limitato.

Per quanto riguarda invece lo sviluppo di software **assembler 8086** (famiglia di microprocessori su cui sono basati i PC-IBM compatibili) è sufficiente disporre di un PC ed utilizzare un assemblatore (come il MacroASM Microsoft o il TurboASM Borland) ed il programma DEBUG del DOS. Con tali strumenti sarà però possibile solamente lo sviluppo di software a livello assembler e non di sistemi HW/SW, tranne che si intenda utilizzare strettamente l'hardware standard presente sui PC. A tale proposito, cioè l'utilizzo anche a basso livello dei PC-IBM compatibili, sarà eventualmente disponibile più avanti del materiale riguardante l'assembler 8086, il DOS e l'utilizzo all'interno del Turbo Pascal Borland delle risorse Hw/Sw della macchina.

## PROCESSO DI SVILUPPO DEL SOFTWARE

### ➔ *ANALISI DEL PROBLEMA ED IDEAZIONE DELLA SOLUZIONE*

Il primo punto da affrontare (relativamente indipendente dall'attrezzatura di laboratorio disponibile) consiste nell'esaminare il problema da affrontare svolgendo un'analisi delle possibili soluzioni. E` indispensabile aver chiaro l'obiettivo da raggiungere nello svolgimento del progetto (sia che si tratti di un'esperienza didattica che di un'applicazione professionale). Innanzitutto occorre sapere se occorre utilizzare un sistema di sviluppo Hw/Sw (nanocomputer o emulatore) oppure solo software (simulatore o PC). Nel primo caso, è necessario individuare la parte hardware da utilizzare, che può comprendere sia sottosistemi già disponibili (come ad esempio le schede Deneb presenti in laboratorio) che sottosistemi "esterni" da progettare e realizzare appositamente (come ad esempio semplici dispositivi di I/O come banchi di switch o diodi led). Progettato l'hardware aggiuntivo specifico pensando anche al software che l'utilizzerà, si passa alla sua realizzazione fisica, che può avvenire parallelamente all'avanzamento della soluzione software. Il progetto del software (che generalmente richiede l'utilizzo di strumenti di pianificazione del lavoro come i diagrammi di flusso) dovrà avvenire tenendo a mente il più possibile i requisiti di modularità e quindi di flessibilità della soluzione. In particolare i concetti di programmazione strutturata e sviluppo TOP-DOWN sono comunque validi anche a livello assembly con le dovute differenze. L'uso di procedure (subroutine) è frequentemente utile per ottenere leggibilità e modificabilità della soluzione. In ogni caso occorre sempre considerare che il tempo dedicato alla definizione rigorosa dell'oggetto che si desidera produrre è tempo utilmente investito nell'economia generale del progetto.

ATTENZIONE: **userete anche ambienti non-IDE** (IDE=integrated Development Environment come ad esempio l'ambiente di sviluppo del Turbo Pascal che contiene al suo interno Editor, Compilatore e Debugger) in modo da riuscire ad apprezzare la comodità degli IDE abituali.
Oggi peraltro si dispone frequentemente di ambienti detti RAD (RAD= Rapid Application Development), soprattutto per lo sviluppo di SW ad alto livello (Object Oriented + Event + Visual Programming come Delphi) che sono ovviamente anche degli IDE.
Nel caso di sviluppo di SW HLL è anche possibile l'integrazione di codice sorgente di vari linguaggi, fra cui l'assembly (ovviamente solo se il Target del codice oggetto è quello dell'assembly che si sta inserendo); ad esempio è possibile inserire subroutine Assembly x86 in Turbo Pascal con la direttiva ASM.

### ➔ *EDITING DEL PROGRAMMA*

Consiste nella stesura del programma sul computer, facendo uso di un Editor (Es Blocconote, Editor del Dos [EDIT], ecc.) che salvi il codice in formato ASCII. Il programma scritto in linguaggio simbolico (assembly), detto file sorgente, deve essere dotato di un'estensione identificabile dall'assemblatore in uso (.SRC per lo Z80 ; .ASM per x86 ; etc.). Ovviamente anche i sorgenti dei linguaggi HLL prevedono proprie estensioni (es. *.PAS oppure *.C).

**Convenzioni da rispettare nella fase di editing (con riferimento particolare agli assemblatori)**

E' consigliato far precedere il listato sorgente da alcune righe di commento (obbligatoriamente precedute dal punto e virgola) che indichi la funzione svolta da tale programma. Iniziare il sorgente indicando la locazione di memoria a partire dalla quale deve essere caricato, mediante la pseudoistruzione ORG. L'elemento terminale del programma "fisico", cioè lo statement da scrivere "più in basso", deve essere la pseudoistruzione END. Prevedendo di interrompere (in fase di debug) l'esecuzione del programma con dei breakpoints, è utile porre una NOP nei punti successivi l'ultima istruzione che si desidera venga eseguita (non è però necessario farlo, chiarisce soltanto maggiormente dove inserire poi i breakpoints). Le etichette o label (utilizzare possibilmente nomi significativi) vanno obbligatoriamente poste a partire dalla prima colonna e devono essere brevi (massimo 8 caratteri), diversamente vengono interpretate come istruzioni. Le istruzioni è utile vengano poste a partire dal decimo carattere in poi ed incolonnate (utilizzare a tale scopo il tasto TAB). I commenti devono essere preceduti dal punto e virgola ed incolonnati a destra delle istruzioni. E` estremamente consigliabile inserire commenti ovunque chiariscano maggiormente il listato (ad esempio spiegando l'utilizzo dei registri usati) ed utilizzare notazioni sintetiche, in modo che tutti gli elementi eventualmente presenti su di una riga (<label>, istruzione, <commento>) non occupino più di 70/75 caratteri. È consigliabile collocare il listato senza saltare righe tranne quando vi è un salto di organizzazione in memoria o si tratta di routine distinte. Affinché i numeri siano interpretati come esadecimali vanno fatti seguire dal carattere "h". Tutti i numeri in questo formato che iniziano con lettere (A/B/C/D/E/F) devono essere preceduti dal numero 0 (ES : 0FFh) per evitare che l'assemblatore li confonda con istruzioni Mnemoniche o Registri interni del MP.

## ➔ *ASSEMBLAGGIO DEL PROGRAMMA*

La fase di assemblaggio di un programma sorgente consiste nella traduzione dei suoi statement in linguaggio macchina. Per questo motivo il nuovo file prodotto, detto modulo oggetto, assume un formato binario. L'assemblatore in pratica svolge la funzione tipica di un compilatore, solo che la traduzione da svolgere è molto semplice, vista la corrispondenza uno a uno tra istruzioni (statement) assembly e istruzioni in linguaggio macchina. In genere gli assemblatori operano in due passate: nella prima scorrono il file sorgente e assegnano un indirizzo a tutti i simboli (label, etc.) presenti nel sorgente (creazione della tabella dei simboli); nella seconda effettuano la traduzione vera e propria tra nomi mnemonici e codici binari delle istruzioni, sostituendo tutti i simboli con l'indirizzo associato nel corso della prima passata. L'assemblatore da noi utilizzato in laboratorio si chiama AZ80.EXE ed è un cross-assembler, visto che "gira" su PC (che hanno processori diversi dallo Z80). Esistono due tipi fondamentali di assemblatori: quelli assoluti e quelli rilocabili. Gli assemblatori assoluti operano una sostituzione di ciascuna etichetta con l'indirizzo assoluto e forniscono la locazione di memoria a partire dalla quale il programma deve essere caricato. I moduli così prodotti non necessitano il più delle volte di essere linkati. Gli assemblatori rilocabili, invece, sia che il sorgente faccia uso di salti relativi che assoluti, mantengono gli indirizzi in forma parametrica, necessitando quindi di una successiva fase di traduzione prima del caricamento. In fase di linking del file oggetto il linker scrive un indirizzo proprio a partire dal quale il loader dovrà caricare in memoria il programma. Se il sorgente fa uso di salti assoluti occorre che il loader rispetti l'indirizzo indicato, diversamente si può caricare il programma a partire da qualsiasi locazione (cioè i salti relativi non necessitano - ovviamente – di rilocazione) !

Esempio Operazioni da effettuare per assemblare il programma (rif. uso assemblatore Z80):
a) lanciare il file batch (che avvia l'assemblatore) seguito dal nome del sorgente senza estensione;
b) osservare il messaggio sul monitor: se non vi sono stati errori (sintattici) si può proseguire, diversamente occorre editare nuovamente il sorgente per correggerlo.
c) il risultato dell'assemblaggio consta di due file: il modulo OBJ che è il file (poi utilizzato dal linker) contenente la traduzione esatta del programma in linguaggio macchina; il modulo LST che è un file contenente informazioni di riepilogo sul risultato dell'assemblaggio come il n° d'ordine di ciascuno statement, la locazione di memoria a partire dalla quale viene caricato, codici esadecimali del linguaggio macchina, gli statement assembly, la tabella dei simboli, etc.
d) è consigliabile effettuare una stampa (o una copia) del file .LST che permetterà poi in fase di debug di inserire dei breakpoints agli indirizzi appropriati.

## ➔ *LINKING DEL PROGRAMMA*

La fase di linking svolge una funzione indispensabile solo quando il programma complessivo è frazionato su più moduli oggetto, per cui è possibile a volte "bypassarla" inserendola entro (in coda a) la fase di assemblaggio. La fase di linking è infatti tipica degli assemblatori rilocabili, e consiste nella fusione del programma oggetto con le proprie procedure e con altri moduli o sottoprogrammi a loro volta rilocabili. Al termine della fase di linking, che conduce alla creazione di un modulo in formato esadecimale dotato di indirizzo di partenza, tutti i parametri vengono sostituiti con i loro valori assoluti ed il file risultante in formato esadecimale è pronto ad essere caricato nell'ambiente di esecuzione.

Operazioni da effettuare per il linking del programma (rif. uso linker Z80):
a) lanciare il file batch (che manda in esecuzione il linker) seguito dal nome del file oggetto (.OBJ) senza estensione;
b) osservare il messaggio sul monitor: se non vi sono stati errori (non dovrebbero esservi avendo già passato la fase di assemblaggio nella quale vengono rilevati gli errori di sintassi) si può proseguire, oppure occorre "tornare indietro".
c) il risultato del linking consta di un file in formato Intel che è pronto da caricare nell'ambiente di esecuzione. Il formato Intel prevede che i caratteri esadecimali che rappresentano i byte siano descritti sotto forma di caratteri ASCII e suddivisi su più righe, ciascuna indicante l'indirizzo di allocazione in memoria ed il numero di byte. L'ultima riga contiene anche dei codici di controllo per la correttezza della trasmissione (secondo la tecnica checksum).

## ➔ *LOADING DEL PROGRAMMA*

Ora il modulo è pronto per essere caricato nella memoria dell'ambiente di esecuzione (ovvero di simulazione o emulazione), per essere "debuggato" ed eseguito. Se si utilizza un sistema di sviluppo con hardware esterno (come l'emulatore) e se non lo si è ancora fatto, occorre interfacciarlo fisicamente al PC (tramite porta seriale). Avviato l'ambiente di emulazione, per caricare il programma in memoria, occorre digitare il comando di lettura del file Intel entro l'ambiente. Giunti a questo punto è possibile controllare da PC l'esecuzione dell'applicazione sviluppata, grazie ai comandi dell'ambiente di debug. Un listato di questi è generalmente ottenibile digitando H (help) dal prompt dell'ambiente.

N.B.: In ambiente di emulazione/simulazione i numeri sono generalmente già automaticamente interpretati come esadecimali, per cui non si deve porre h dopo di essi.

## ➔ *DEBUGGING DEL PROGRAMMA*

Consiste nella messa a punto del programma, eliminandone gli errori logici (cioè quelli introdotti in fase di ideazione della soluzione e rilevabili soltanto a tempo di esecuzione). Come strumenti di debug sono disponibili lo step by step (esecuzione di un'istruzione per volta) ed i breakpoint (punti d'arresto dell'esecuzione del programma). In entrambi i casi (dopo ogni statement o dove c'è un breakpoint) viene presentata una maschera indicante lo stato interno dei registri e l'istruzione successiva (in linguaggio assembler), inoltre è possibile esaminare il contenuto della memoria (visto che si è al livello del prompt dell'ambiente). Prima di mettere dei breakpoint occorre conoscere le locazioni delle istruzioni interessate, per questo può essere utile disassemblare con un apposito comando la zona di memoria ove è allocato il programma oppure disporre di una stampa del programma dalla quale risultino gli indirizzi delle celle dove sono inseriti i vari statement (.LST). Per esaminare i vari comandi disponibili a questo livello (come ad esempio il <Go> o il <Disassemble>) consultare le dispense o il materiale riguardante i sistema di sviluppo utilizzato di volta in volta.

N.B.: Solo dopo aver accuratamente verificato il corretto funzionamento del programma con varie istanze di dati in ingresso è possibile affermare che questo è corretto. E` necessario osservare che molti errori logici sono estremamente subdoli perchè avvengono solo alcune volte in concomitanza di particolari valori di input o di particolari istanti di utilizzo. Un programma molto spesso appare funzionante anche se in realtà non lo è, contenendo vari errori logici e funzionando in modo errato anche solo una volta su un milione. Nel caso si rilevino errori occorre ripetere il processo di sviluppo del software ciclicamente fino a che il test in fase di debug non è completamente soddisfacente.

## ➔ *ESECUZIONE DEL PROGRAMMA*

Questa fase è in pratica coincidente con quella precedente, differendo nel caso l'applicazione sia da eseguire su una scheda autonoma, nel qual caso occorre in genere inserire il programma su EPROM, sostituire l'I.C.E. dell'emulatore con un vero microprocessore e inserire nella scheda la EPROM contenente il software prodotto. A questo punto l'applicazione hardware/software è completata ed utilizzabile.

*Questa dispensa è una revisione di materiale che avevo prodotto in A.S. precedenti e fa riferimento ad ambienti di sviluppo che operavano in ambiente DOS, SO per il quale Windows 7, da quest'anno installato sui PC della scuola , risulta non retrocompatibile.*
*Per questo motivo risulterà necessario utilizzare ambienti di virtualizzazione di altri SO precedenti, come Virtualbox che già usate per Windows XP dove "gira" Turbo Pascal o altri SW come DOSBOX.*
*Mi scuso per i disagi dovuti a tutti questi passaggi ma ritengo che l'utilizzo di (scomodi!!!) ambienti a riga di comando (interfaccia CLI Command Line Interface) , non-IDE e non RAD, di Virtualizzatori faccia parte delle conoscenze che un informatico deve possedere, anche se poi, in caso di utilizzo "professionale" verranno scelti degli ambinti di sviluppo più efficienti.*

*Longanesi Claudio* - revisione dicembre 2013