Laboratorio di Linguaggi di Sistema – a.a. 2006/2007 – Progetto finale I (base)

Il problema

Si vuole realizzare un *simulatore di CPU*, ovvero un interprete di programmi in linguaggio Assembler. Il programma deve simulare una memoria di 64Kb, caricare in questa memoria dei moduli eseguibili binari, come specificato sulla riga di comando, iniziare l'esecuzione del codice caricato (partendo da una locazione specificata sulla riga di comando), e alla fine dell'esecuzione stampare il contenuto di un'area di memoria specificata dall'utente sulla riga di comando.

Sintassi e parametri

Il simulatore deve avere la seguente sintassi su riga di comando:

$$\texttt{llscpu } mod_1\texttt{@}addr_1 \texttt{ ... } mod_n\texttt{@}addr_n \; addr_{start} \texttt{ [} addr_{dump} \texttt{ len]}$$

Il nome del comando (llscpu) è seguito da una lista di *moduli binari* da caricare; ogni modulo è identificato dal nome del file che contiene il codice, seguito da un simbolo "@" e da un indirizzo (entro i 64Kb di memoria simulata) che indica la locazione iniziale da cui caricare il modulo. Il primo parametro che non rappresenta un modulo è l'indirizzo da cui iniziare l'esecuzione; a questo può opzionalmente seguire l'indicazione di un'area di memoria, indicata da indirizzo iniziale e lunghezza, il cui contenuto deve essere stampato, byte per byte, in esadecimale, alla fine dell'esecuzione. Tutti gli indirizzi e la lunghezza dell'area da stampare possono essere espressi indifferentemente in decimale, in esadecimale (preceduti da 0x) o in ottale (preceduti da 0), come in C.

Formato dei moduli binari

I moduli binari hanno il seguente formato di file:

module length (*m*)	*m* bytes			
	header length (*h*)	header data	executable code	checksum
4 bytes	4 bytes	*h* bytes	*m*-4-*h* bytes	4 bytes

Il file consiste dunque di un long (4 byte) indicante la lunghezza totale del modulo (esclusi i 4 byte iniziali), seguito da un blocco di header, a sua volta composto di un analogo campo lunghezza di 4 byte, e di tanti byte di header quanto indicato nel campo lunghezza. All'header seguono i dati binari veri e propri (che dovranno essere caricati in memoria); termina il file un checksum di 4 byte, ottenuto sommando fra di loro i valori di tutti i byte che compongono il modulo, inclusi lunghezze ed header, ma escluso il checksum stesso. Il simulatore può ignorare i dati dell'header, ma deve verificare la validità del checksum prima di accettare un modulo (e stampare un messaggio d'errore se il checksum è invalido).

La CPU simulata

La CPU simulata è un semplice processore a 8 bit, dotato di 3 registri (a 8 bit) denominati A (accumulatore), X (indice) e S (stack pointer); la CPU dispone inoltre di un registro interno PC (program counter) a 16 bit, che in ogni istante punta alla *prossima istruzione* da eseguire, e di un registro di stato SR (status register) contenenti alcuni bit che memorizzano il risultato dei confronti (usati poi per le istruzioni di salto condizionato). Le istruzioni sono tutte codificate da opcode di 1 byte, e possono avere un parametro nel byte immediatamente seguente. L'indirizzamento "naturale" è al byte (le prestazioni non sono influenzate da allineamenti a granularità più grossa, come avviene invece sui processori a 16 o 32 bit). Tutti gli indirizzamenti di memoria sono relativi al PC corrente. Gli opcode hanno una organizzazione regolare, riassunta nella seguente tabella:

bit 7-6	bit 5	bit 4-1	bit 0
Registro su cui operare: 01 = A 10 = X 11 = S	Indirizzamento: 0 = semplice 1= indicizzato da X	codice operativo: 0000 = LD (load) 0001 = ST (store) 0010 = ADD 0011 = SUB 0100 = SHL (shift left) 0101 = SHR (shift right) 0110 = ROL (rotate left) 0111 = ROR (rotate right) 1000 = PUSH 1001 = POP 1010 = TX (transfer X to ...) 1011 = TS (transfer S to ...) 1100 = TA (transfer A to ...) 1101 = CMP (compare)	indirezione degli argomenti: 0 = valore nel byte successivo 1 = valore nella cella di memoria con offset (relativo al PC) pari al valore contenuto nel byte successivo, interpretato come intero con segno

00 = opcode estesi		0000 = NOP (no operation) 0001 = JMP (jump) 0010 = JSR (jump subroutine) 0011 = RTS (return from subroutine) 0100 = JEQ (jump if equal) 0101 = JLT (jump if less than) 0110 = JGT (jump if greater than) 0111 = JNE (jump if not equal) 1000 = STOP (stop execution)	

Ovviamente, non tutte le combinazioni di bit hanno senso. Alcuni esempi:

valori in memoria		***mnemonico***	***semantica***
64 33	01 0 000 0 00100001	LDA #33	carica nel registro A il valore assoluto 33
65 33	01 0 000 1 00100001	LDA @33	carica nel registro A il valore contenuto nella cella di memoria di indirizzo PC+33
198 6	11 0 0011 0 00000110	SUBS #6	sottrae il valore assoluto 6 al contenuto corrente dello stack pointer
84	01 0 1010 0	TXA	copia il valore nel registro X in A
123 12	01 1 1101 1 00001100	CMPA @12+X	confronta il valore contenuto nel registro A con quello contenuto nella cella di memoria situata all'indirizzo PC+X+12
9 244	00 0 0100 1 11110100	JEQ @-12	se il risultato dell'ultimo CMP effettuato indica che i valori comparati erano identici, continua l'esecuzione dalla locazione di indirizzo PC-12
37 10	00 1 0010 1 00001010	JSR @10+X	salva il PC corrente (due byte) sullo stack (nella cella puntata da S), decrementa S di conseguenza, e continua l'esecuzione dalla locazione di indirizzo PC+10+X
4	00 0 0010 0	RTS	recupera il PC (due byte) dallo stack (nella cella puntata da S), incrementa S di conseguenza, e continua l'esecuzione dalla locazione puntata dal nuovo PC
16	00 0 1000 0	STOP	termina l'esecuzione, arresta il simulatore (opzionalmente, stampa la memoria come richiesto)

Per quanto riguarda i punti non illustrati dagli esempi, nonché per i dettagli implementativi (es.: disposizione dei bit all'interno dello SR), lo studente è libero di adottare le scelte (consistenti e ragionevoli) ritenute più opportune.

Modalità di consegna

L'elaborato deve essere consegnato **improrogabilmente** entro il 22 gennaio 2007 alle 20:00, e deve essere costituito da una stampa del codice sorgente, adeguatamente formattato e commentato, da una stampa con esempi di esecuzione, e da una breve relazione scritta (2-3 pagine) che descrive il progetto stesso. La versione a stampa può essere consegnata direttamente al docente, oppure depositata presso il centralino del Dipartimento di Informatica. Tutto il materiale deve essere anche inviato via email al docente (gervasi@di.unipi.it), in forma elettronica, entro il termine indicato.

Altre informazioni

Il programma deve gestire correttamente le situazioni eccezionali, quali ad esempio mancanza di memoria, impossibilità di accedere a certi file in quanto non esistenti o non dotati dei diritti necessari all'operazione, ecc. Gli eventuali errori devono essere segnalati all'utente attraverso messaggi su *standard error*. In nessun caso il programma deve inviare su *standard output* dati diversi da quelli richiesti.

Esempi

Verranno pubblicati sul sito del corso alcuni moduli eseguibili pronti per l'esecuzione, con i risultati attesi.