Politecnico di Milano - Dipartimento di Elettronica e informazione

Prof. Mauro Negri

Fondamenti di Informatica **1 marzo 2012**

I appello

Matricola_____ Cognome _____ Nome _____

Durata prova: 2 ore

Istruzioni

LE RISPOSTE DEVONO ESSERE SCRITTE SU QUESTO PLICO (anche sul retro)

Non separare questi fogli. Si può utilizzare la matita.

L'uso di cellulari, libri, eserciziari, appunti o calcolatrici durante lo svolgimento della prova comporta l'annullamento della prova.

Non è possibile uscire dall'aula durante la prova tranne in casi eccezionali.

Esercizio 1 (5 punti) _____

Esercizio 2 (4 punti) _____

Esercizio 3 (7 punti) _____

Esercizio 4 (8 punti) _____

Esercizio 5 (6 punti) _____

Punteggio totale (30 punti) _____

Esercizio 1.

1. Indicare la rappresentazione in complemento a 2, utilizzando 8 bit, dei numeri:
   A = 100 (base 10) 0110 010
   B = -87 (base 10) 0101 0111 -> 1010 1001

2. Eseguire la somma A+B in binario dei numeri interi A=170 e B=150, utilizzando 9 bit per la rappresentazione. Mostrare il risultato in binario generato dall'unità aritmetico-logica; riportare inoltre il corrispondente valore in base 10, il valore del bit di carry (riporto) e di overflow dopo l'operazione, indicando in breve il perché del valore di carry e overflow.

Range valori da -256 a +255

010101010 170
010010110 150

101000000

Risultato (base 10): -192
bit carry: 0
bit overflow: 1

Motivazione:
Carry non esiste riporto oltre il nono bit
Overflow esiste perché primo bit è concorde negli operandi e con il risultato

3. Data la definizione della variabile float v= – 88,1 descrivere la sua rappresentazione in virgola mobile in base allo standard IEEE754 semplice precisione FP32 descritto a lezione;
   a) Definire prima il numero nel formato normalizzato intermedio (TRONCARE parte decimale alla terza cifra decimale):

$$-1{,}376 * 2^{6}$$

   b) Codificare poi il numero normalizzato usando il formato FP32:
      S (1 bit di segno): 1
      M (23 bit per la mantissa – fermarsi ai primi 6 bit): 011000
      E (8 bit per l'esponente): 10000101

   c) Indicare quale sia il vero valore (in base 10) descritto dalla sequenza di bit memorizzata in S, M ed E del punto b) precedente (si consiglia di esprimere la mantissa in termini frazionari).

-88

4. Data la rappresentazione IEEE754 - FP32 del numero 128,0 indicare quale sia il primo numero decimale (esprimerlo eventualmente come formula) rappresentabile successivo al numero 128,0.

$$\mathbf{128 + 2^{-23} * 2^{7}}$$

Esercizio 2. Tradurre il seguente programma C in assembler (ipotizzare che lo stack sia già stato allocato e SP già inizializzato.

```
#include <stdio.h>
struct temp {int a; int b;};
struct temp T; int c,d;
int fun(struct temp t, int *a)
    {int vet[2];vet[0]=*a; return vet[0];}
int main ()
    { …
      c=fun(T, &d);
      }
```

Istruzioni assembly

[W:] ADD oper1,oper2	somma in oper2
[W:] MOV oper1, oper2	copia in oper2
[W:] RTS	ritorno da funzione
[W:] JTS oper	invocazione sottoprogramma
[W:] EXIT	termina esecuzione programma

Modalità di indirizzamento operandi delle istruzioni:

- Ri o SP contenuto del registro Ri o SP
- (Ri) o (SP) contenuto della parola di memoria il cui indirizzo è nel registro Ri o SP
- #X valore del simbolo X

Ipotesi: il contenuto di R1,…, R6 (non R0 e SP) non va salvato prima dell'uso dei registri.

```
TA: .RES 1
TB: .RES 1
C: .RES 1
D: .RES 1
MAIN: ADD #-1, SP
      MOV TA, (SP)
      ADD #-1, SP
      MOV TB, (SP)
      ADD #-1, SP
      MOV #D, (SP)
      ADD #-1, SP
      JTS FUN
      ADD #4, SP
      MOV (SP), C
      EXIT
FUN: MOV R0, (SP)
      ADD #-1, SP
      MOV SP, R0
      ADD#-2,SP

      MOV R0, R1
      ADD #3, R1
      MOV (R1), R2
      MOV (R2), (R0)

      MOV R0,R1
      ADD #6,R1
      MOV (R0), (R1)

      ADD # 3, SP
      MOV (SP), R0
      RTS
```

Esercizio 3 Date le seguenti definizioni

#define max 20
int vet[max]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
struct EL {int base; int esponente; struct EL * prossimo;};
struct EL * LIS;

Scrivere la funzione C chiamata **Modifica** che dichiara due matrici bidimensionali **BASE[**max, 10] di numeri interi e **ESP**[max, 10] di numeri naturali (non scrivere le istruzioni per il caricamento del contenuto delle matrici – si ipotizzi nel seguito che le matrici siano già state caricate). La funzione riceve come parametro il vettore **vet** e procede scandendo le due matrici per righe dalla prima all'ultima e genera la somma degli elementi della riga i-esima di **BASE** e la somma degli elementi della riga i-esima di **ESP;** se la somma di **ESP** non è compresa tra 0 e 19 non si fa nulla e si passa ad eseguire le somme della riga successiva, altrimenti le due somme sono usate per aggiornare il vettore **vet** nel seguente modo:

Per ogni i ∈ [1,20]:

$$\text{Vet}\left[\sum_{s=1}^{10} \text{ESP}[i,s]\right] = \sum_{s=1}^{10} \text{BASE}[i,s]$$

La funzione infine ritorna come parametro il vettore modificato **vet**.

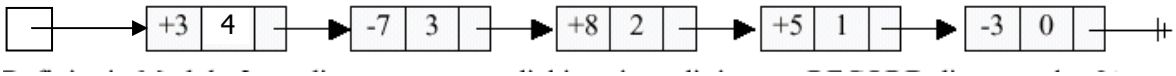
```
#define COL 10
void modifica (int vet[]){
     int i, j;
     int bas [MAX][COL], esp [MAX][COL];
     int sommaB, sommaE;

…….
     for (i=0; i<MAX; i++){
          sommaB = 0;   sommaE = 0;
          for (j=0; j<COL; j++){sommaB += bas[i][j];
                                sommaE += esp[i][j];
                               }
//printf non richieste
           printf ("\nSomma base riga %d = %d", i+1, sommaB);
           printf ("\nSomma esp riga %d = %d", i+1, sommaE);

          if (sommaE>=0 && sommaE<MAX)
              {vet[sommaE] = sommaB;
               printf("\nvett[%d] = %d", sommaE, sommaB);
              }
     }
}
```

Esercizio 4. Scrivere la funzione C chiamata **INS** che riceve il vettore **vet** e la lista **LIS** come parametro. La funzione verifica che la lista sia vuota altrimenti finisce l'esecuzione e ritorna 0. Per ogni elemento del vettore genera un elemento dinamico (se la generazione fallisce cancella la lista creata sino a quel punto e ritorna 0) e carica rispettivamente la posizione dell'elemento corrente del vettore nel campo **esponente** e il valore contenuto nella posizione corrente del vettore nel campo **base** dell'elemento dinamico. la funzione inserisce l'elemento dinamico in fondo alla lista; si noti di scandire il vettore in modo che la lista alla fine risulti ordinata per valori decrescenti del campo **esponente**.

La seguente figura mostra un esempio dello stato della lista nella quale il primo campo rappresenta la base e il secondo l'esponente.



La funzione infine ritorna la lista creata e il numero 1 come parametri.

```
int ins (int vet[], struct el **lista)
{
     struct el *temp, *prec;
     int i;

     if (*lista != NULL)        return 0;

     for (i=MAX-1;i>=0;i--)
       // per ottenere l'ordinamento desiderato inserendo in coda
        { temp = (struct el *) malloc (sizeof(struct el)*1);

          if (temp==NULL) {svuotaLista (lista); return 0;}

          temp->base = vet[i];
          temp->esp = i;
          temp->next=NULL;

          if (*lista==NULL) //inserisco il primo
               {*lista = temp;
                prec = temp; // per non ricalcolare ultimo ogni volta
               }
          else{prec->next = temp; prec = prec->next;}
        }
     return 1;
}
void svuotaLista(struct el **lista)
{    struct el * temp;
     while (*lista!=NULL)
     {    temp = *lista;
          *lista = (*lista)->next;
          free(temp);
     }
}
```

Esercizio 5. Scrivere la funzione C chiamata **calcola** che riceve come parametri un intero **E** e la lista **LIS** e verifica che **E** appartenga all'intervallo [0,19] (in caso negativo termina l'esecuzione e ritorna il codice -1). Poi procede a cercare l'elemento della lista che ha nel campo **esponente** il valore **E** (se non lo trova termina l'esecuzione e ritorna il codice -1). Quando lo trova esegue il calcolo della potenza $\mathbf{B}^{\mathbf{E}}$ con **B** corrispondente al valore del campo **base** dell'elemento dinamico individuato. Restituisce infine come parametri la potenza calcolata e il codice 1, ad eccezione del caso in cui base ed esponente siano 0 per il quale la funzione ritorna il codice -2 e nessun risultato.

Attenzione. L'implementazione della funzione **DEVE** utilizzare la seguente funzione ***potenza*** **(da invocare, ma non da implementare)** che riceve base (intero) ed esponente (numero naturale) come parametri e restituisce sempre come parametro il risultato dell'espressione $|\mathbf{base}|^{\mathbf{esponente}}$ (la funzione non considera quindi il segno della base e pertanto -2^3 è calcolato come 2^3 e la funzione ritorna +8); si noti inoltre che la funzione non accetta base ed esponente uguali a 0.

```
int calcola (int val, struct el *lista, int *result)
{    struct el *temp, *prec;   int i;

     if (val>=MAX || val<0)return -1;

// scansione che non ipotizza che esista l'esponente cercato
// in ogni caso serve everificare se la lista sia vuota

     while (lista!=NULL && lista->esp!=val)lista = lista->next;

     if (lista == NULL)
          {// non trovato
           return -1;
          }

//0 elevato a 0 da verificare prima dell'esecuzione della funzione potenza
// perché la funzione non li acetta

     if (lista->base==0 && lista->esp==0)return -2;

     *result = potenza(lista->base, lista->esp);
// aggiustamento per base negativa
     if (lista->base<0 && lista->esp%2!=0) *result = - *result;
     return 1;
}
```