Nome: _____ Cognome: _____

Matricola: _____ Anno di immatricolazione _____

Prova di laboratorio di Architettura degli elaboratori

Istruzioni: *Ogni esercizio é descritto tramite un programma in linguaggio C. Se la realizzazione di un I/O non é richiesta, le variabili possono essere inizializzate istruzioni assembler o con direttive (nel caso degli array). Si raccomanda di seguire le convenzioni del linguaggio assembler specie per ció che riguarda le funzioni. Si raccomanda di utilizzare i commenti per indicare la corrispondenza fra variabili del C e registri. I commenti sono anche utili per descrivere cosa volete fare nel codice.*

1. [1.0] Programma che calcola il prodotto di due numeri naturali in una CPU a 32 bit priva di moltiplicatore. L'esercizio deve essere risolto senza utilizzare macro (`bge ....`). Si ignorino eventuali problemi di overflow.

```
main()
{
  unsigned int x, y; /* moltiplicando e moltiplicatore */
  unsigned int p; /* prodotto */
  int i, tmp; /* variabili */

  x=16; y=18; p=0;
  /* acquisisce x,y (non importa farlo) */
  i=0;
  while (i<32)
    {
      tmp=y & 1;   /* i-th bit of y */
      if (tmp!=0)
        p=p+x;
      y=y>>1;
      x=x<<1;       /* x=x*2 */
      i=i+1;
    }
  /* stampa p (non importa farlo) */
}
```

Soluzione

```
.text

addi $s0, $zero, 1024 # x
addi $s1, $zero, 16 # y
addi $s2, $zero, 0  # p

addi $t0, $zero, 0
addi $t1, $zero, 32
```

Nome: _____ Cognome: _____

```
loop: beq $t0, $t1, endloop
  andi $t2, $s1, 1
  beq $t2, $zero, label
    add $s2, $s2, $s0
  label: srl $s1, $s1, 1
  sll $s0, $s0, 1
  addi $t0, $t0, 1
  j loop
endloop:
```

2. [2.0] Verifica dell'ordinamento di un vettore. Anche in questo caso non si devono utilizzare macro. Il codice non é ottimizzato, ma questo non riguarda l'esercizio.

```
main()
{
  int array[8]={0,1,4,2,7,8,4,6};
  int i;

  int ord_c, ord_sc; /* ordinamento crescente e str. crescente */

  i=0;
  ord_c=ord_sc=1; /* true */
  while (i<7)
    {
       if (array[i]>=array[i+1])
           ord_sc=0;
       if (array[i]>array[i+1])
           ord_c=0;
       i=i+1;
    }
  /* stampa il tipo di ordinamento (da non fare) */
}
```

Soluzione

```
.data

array: .word 0, 1, 2, 4, 6, 7, 9, 0

.text
addi $s0, $zero, 1 # ord_c
addi $s1, $zero, 1 # ord_sc

addi $s2, $zero, 0 # i
addi $t0, $zero, 7
```

Nome: _____ Cognome: _____

```
addi $t1, $zero, 0

# loop non ottimizzato
lw $t2, array($t1)
loop: beq $s2, $t0, endloop
  addi $s2, $s2, 1
  sll $t1, $s2, 2 # addr=4*i
  lw $t3, array($t1)
  slt $t4, $t2, $t3 # array[i+1]<array[i]
  bne $t4, $zero, label0
    addi $s1, $zero, 0
  label0: slt $t4, $t3, $t2 # array[i]>array[i+1]
    beq $t4, $zero, label1
    addi $s0, $zero, 0
  label1: addi $t2, $t3, 0 # evita una lw per ciclo
  j loop
endloop:
```

3. [2.0] Funzione che calcola un esempio di espressione. Si faccia l'ipotesi di dover preservare tutti i registri utilizzati dall'espressione (compreso i `$t`).

```
int main()
{
  int a,b,c,d;
  int v;

  a=7; b=4; c=4; d=2;
  ....
  v=dist(a,b,c,d);
}

int dist(int a, int b, int c, int d)
{
  int result;

  result=(a+b)>>(c-d)+b<<d;
  /* << e >> sono prioritari rispetto alla somma */
  return result;
}
```

Soluzione

```
.text

main:
  addi $s0, $zero, 7 # a
```

Nome: _____ Cognome: _____

```
  addi $s1, $zero, 4 # b
  addi $s2, $zero, 4 # c
  addi $s3, $zero, 2 # d

# possible operations on $s0..3 and other registers

  addi $a0, $s0, 0    # argument 1
  addi $a1, $s1, 0    # argument 2
  addi $a2, $s2, 0    # argument 3
  addi $a3, $s3, 0    # argument 4
  jal  dist    # call Function
  addi  $s4, $v0, 0   # returned value

#
dist:
  addi $sp, $sp, -12  # make space on stack to
                      # store three registers
  sw  $s0, 0($sp)    # save $s0 on stack
  sw  $s1, 4($sp)    # save $s1 on stack
  sw  $s2, 8($sp)    # save $s2 on stack

  add $s0, $a1, $a0  # a+b
  sub $s1, $a2, $a3  # c-d
  sllv $s2, $a1, $a3  # b<<d
  srlv $s0, $s0, $s1  # >>
  add $v0, $s0, $s2
  lw $s0, 0($sp)     # restore $s0 from stack
  lw $s1, 4($sp)     # restore $t0 from stack
  lw $s2, 8($sp)     # restore $t0 from stack
  addi $sp, $sp, 12   # deallocate stack space
  jr  $ra            # return to caller
```

4. [1.5] Programma inteso a verificare le differenze fra & ed && in C.

```
int main()
{
  int x, y, w;

  x=9; y=6;
  w=0;
  if (x & y)       /* bitwise and */
    w=1;
  else
    if (x && y)      /* logical and */
      w=2;
}
```

Nome: _____ Cognome: _____

Soluzione

```
.text

addi $s0, $zero, 9
addi $s1, $zero, 7

addi $s2, $zero, 0

# nota: in base 2 ho 1001 e 0110 sono entrambi != 0
# e quindi "veri" per && ma il loro and bit a bit da 0
# ovvero falso

and $t0, $s0, $s1

beq $t0, $zero, label
addi $s2, $zero, 1
j join
label: beq $s0, $zero, join
beq $s1, $zero, join
addi $s2, $zero, 2
join:
```