

# Contents

|                                              |          |
|----------------------------------------------|----------|
| <b>Esercitazione</b>                         | <b>1</b> |
| 23/09/20 . . . . .                           | 1        |
| “eserciziAssemblaggio” esercizio 1 . . . . . | 1        |
| 25/09/20 . . . . .                           | 3        |
| “eserciziAssemblaggio” esercizio 2 . . . . . | 3        |
| 30/09/20 . . . . .                           | 4        |
| “eserciziAssembler1” esercizio 9 . . . . .   | 4        |
| “eserciziAssembler1” esercizio 10 . . . . .  | 6        |
| 07/10/20 . . . . .                           | 8        |
| “eserciziAssembler2” esercizio 3 . . . . .   | 8        |
| “LogicaCombinatoria” esercizio 11 . . . . .  | 10       |
| “LogicaSequenziale” esercizio 3 . . . . .    | 11       |
| “LogicaSequenziale” esercizio 4 . . . . .    | 11       |
| “LogicaSequenziale” esercizio 7 . . . . .    | 12       |

## Esercitazione

23/09/20

“eserciziAssemblaggio” esercizio 1

Dati i seguenti moduli:

- Modulo “main”:

```
.data
STRUCT: .space 20
VECT: .space 12
INT: .int 23

.text
.globl MAIN

MAIN:
    li $t0, 0xFFFF0ABCC
    sw $t0, STRUCT
    lw $t1, VECT
    beq $t0, $t0, MODULE
MAINEND:
    syscall
```

- Modulo “module”:

```
.data
ALPHA: .byte 'Y'

.text
.globl MODULE
RESTART:
    lw $t3, INT
MODULE:
    lb $t4, ALPHA
    sub $t4, $t4, $t3
    beq $t4, $0, RESTART
MODEND:
    j MAINEND
```

Si compilino le quattro tabelle relative a:

1. i moduli oggetto prodotti dall’assemblatore
2. le basi di rilocalizzazione del codice e dei dati dei moduli

3. La tabella globale dei simboli
4. il contenuto del file eseguibile prodotto dal linker

#### 1. Tabella file oggetto

|                         |  |                        |
|-------------------------|--|------------------------|
| dim text: 18            |  | dim text: 14           |
| dim data: 24            |  | dim data: 1            |
|                         |  |                        |
| text:                   |  | text:                  |
| 0 lui \$t0, FFF0        |  | 0 lw \$t3, 0000(\$gp)  |
| 4 ori \$t0, \$t0, ABCC  |  | 4 lb \$t4, 0000(\$gp)  |
| 8 sw \$t0, 0000(\$gp)   |  | 8 sub \$t4, \$t4, \$t3 |
| C lw \$t1, 0000(\$gp)   |  | C beq \$t4, \$0, FFFC  |
| 10 beq \$t0, \$t0, 0000 |  | 10 j 000 0000          |
| 14 syscall              |  |                        |
|                         |  |                        |
| data:                   |  | data:                  |
| 0 uninitialized         |  | 0 0000 0059            |
| 14 uninitialized        |  |                        |
| 20 0000 0017            |  |                        |
|                         |  |                        |
| symbols:                |  | symbols:               |
| STRUCT D 0000 0000      |  | ALPHA D 0000 0000      |
| VECT D 0000 0014        |  | RESTART T 0000 0000    |
| INT D 0000 0020         |  | MODULE T 0000 0004     |
| MAIN T 0000 0000        |  | MODEND T 0000 0010     |
| MAINEND T 0000 0014     |  |                        |
|                         |  |                        |
| relocation:             |  | relocation:            |
| 8 sw STRUCT             |  | 0 lw INT               |
| C lw VECT               |  | 4 lb ALPHA             |
| 10 beq MODULE           |  | 10 j MAINEND           |

#### 2. Basi di rilocalizzazione

|           | main      | module    |
|-----------|-----------|-----------|
| base text | 0040 0000 | 0040 0018 |
| base data | 1000 0000 | 1000 0024 |

#### 3. Tabella globale dei simboli

| simbolo   | valore iniziale | base       | valore finale |
|-----------|-----------------|------------|---------------|
| STRUCT    | 0000 0000       | 1000 0000  | 1000 0000     |
| VECT      | 0000 0014       | 1000 0000  | 1000 0014     |
| INT       | 0000 0020       | 1000 0000  | 1000 0020     |
| MAIN      | 0000 0000       | 0040 0000  | 0040 0000     |
| MAINEND   | 0000 0014       | 0040 0000  | 0040 0014     |
| ALPHA     | 0000 0000       | 1000 0024  | 1000 0024     |
| RESTART   | 0000 0000       | 0040 0018  | 0040 0018     |
| MODULE    | 0000 0004       | 0040 0018  | 0040 001C     |
| MODULEEND | 0000 0010       | 0004 00018 | 0040 0028     |

#### 4. Eseguibile

```

0040 0000 lui $t0, FFF0
0040 0004 ori $t0, $t0, ABCC
0040 0008 sw $t0, 8000($gp)
0040 000C lw $t1, 8014($gp)

```

```

0040 0010 beq $t0, $t0, 0002
0040 0014 syscall
0040 0018 lw $t3, 8020($gp)
0040 001C lb $t4, 8024($gp)
0040 0020 sub $t4, $t4, $t3
0040 0024 beq $t4, $0, FFFC
0040 0028 j 010 0005

```

25/09/20

## “eserciziAssemblaggio” esercizio 2

- Modulo “Main”:

```

.data
INT: .word 37
BLOCK: .space 12

.text
.globl MAIN
MAIN:
    addi $t0, $0, 0x100A
    sw $t0, INT
    la $t1, BLOCK
    lw $t2, ($t1)
    j LIBRARY
MAINEND:
    syscall

```

- Modulo “Library”:

```

.data
VAR: .space 4

.text
.globl LIBRARY
LIBRARY:
    lw $t3, VAR
    beq $t3, $t2, MAINEND
    addi $t3, $t3, 1
LIBEND:
    bne $t3, $t2, LIBRARY
    syscall

```

Si compilino le quattro tabelle relative a:

- i moduli oggetto prodotti dall’assemblatore
- le basi di rilocazione del codice e dei dati dei moduli
- La tabella globale dei simboli
- il contenuto del file eseguibile prodotto dal linker

---

### 1. Tabella file oggetto

|                         |  |                         |
|-------------------------|--|-------------------------|
| dim text: 1C            |  | dim text: 14            |
| dim data: 10            |  | dim data: 4             |
| text:                   |  | text:                   |
| 0 addi \$t0, \$t0, 100A |  | 0 lw \$t3, 0000(\$gp)   |
| 4 sw \$t0, 0000(\$gp)   |  | 4 beq \$t3, \$t2, 0000  |
| 8 lui \$t1, 0000        |  | 8 addi \$t3, \$t2, 0001 |
| C ori \$t1, 0000        |  | C bne \$t3, \$t2, FFFC  |
| 10 lw \$t2, 0000(\$gp)  |  | 10 syscall              |
| 14 j 000 0000           |  |                         |

```

18 syscall |
data:      | data:
0 0000 0025 | 0 uninitialized
4 uninitialized |

symbols:   | symbols:
INT      D 0000 0000 | VAR      D 0000 0000
BLOCK    D 0000 0004 | LIBRARY  T 0000 0000
MAIN     D 0000 0000 | LIBEND   T 0000 000C
MAINEND  T 0000 0018 |

relocation: | relocation:
4 sw INT    | 0 lw VAR
8 lui $hi(BLOCK) | 4 beq MAINEND
C ori %lo(BLOCK) |
14 j LIBRARY |

```

## 2. Basi di rilocazione

|           | main      | module    |
|-----------|-----------|-----------|
| base text | 0040 0000 | 0040 001C |
| base data | 1000 0000 | 1000 0010 |

## 3. Tabella globale dei simboli

| simbolo | valore iniziale | base      | valore finale |
|---------|-----------------|-----------|---------------|
| INT     | 0000 0000       | 1000 0000 | 1000 0000     |
| BLOCK   | 0000 0004       | 1000 0000 | 1000 0004     |
| MAIN    | 0000 0000       | 0040 0000 | 0040 0000     |
| MAINEND | 0000 0018       | 0040 0000 | 0040 0018     |
| VAR     | 0000 0000       | 1000 0010 | 1000 0010     |
| LIBRARY | 0000 0000       | 0040 001C | 0040 001C     |
| LIBEND  | 0000 000C       | 0040 001C | 0040 0022     |

## 4. Eseguitibile

```

0040 0000 addi $t0, $0, 100A
0040 0004 sw $t0, 8000($gp)
0040 0008 lui $t1, 1000
0040 000C ori $t1, $t1, 0004
0040 0010 lw $t2, 0000($t1)
0040 0014 j 010 0007
0040 0018 syscall
0040 001C lw $t3, 8010($gp)
0040 0020 beq $t3, $t2, FFFD
0040 0024 addi $t3, $t3, 0001
0040 0028 bne $t3, $t2, FFFC
0040 002C syscall

```

30/09/20

## “eserciziAssembler1” esercizio 9

Si traduca il seguente programma C in MIPS. Il modello di memoria è quello standard e gli interi sono a 32 bit. Non si eseguano ottimizzazioni. Si facciano le seguenti ipotesi:

- non si usa il frame pointer
- le variabili locali sono allocate nei registri (se possibile)

- vanno salvati solo i registri necessari

Si svolgano i quattro punti:

1. Si traducano in linguaggio MIPS le dichiarazioni globali e si indichi l'indirizzo di ciascuna variabile globale dichiarata
2. Si traducano il linguaggio macchina il codice del programma principale `main`
3. Si descrivano l'area di attivazione della funzione `binary` e l'allocazione delle variabili locali nei registri
4. Si traduca in linguaggio macchina il codice della funzione `binary`

```
#define N 16
int byte = 64;
int elem;
int elem;
int list[N];

int *binary(int i, int val) {
    int *p;
    p = &list[i];
    if (i < 0)
        return list;
    else if (*p == val)
        return p;
    else
        return binary(i / 2 - 1, val + 1);
}

int main(void) {
    elem = *binary(N - 1, byte);
}
```

1. MIPS e indirizzo:

| MIPS            | Indirizzo   |
|-----------------|-------------|
| .data           | NA          |
| .eqv N, 16      | NA          |
| BYTE: .word 64  | 0x1000 0000 |
| ELEM: .word     | 0x1000 0004 |
| LIST: .space 64 | 0x1000 0008 |

2. MIPS relativo a `main`:

```
MAIN:
    li $t0, N
    subi $a0, $t0, 1
    lw $a1, BYTE
    jal BINARY
    lw $t0, ($v0)
    sw $t0, ELEM
```

3. Area di attivazione e registri di `binary`

| contenuto simbolico | offset rispetto a \$sp |
|---------------------|------------------------|
| \$ra                | 4                      |
| \$s0                | 0                      |

| Parametro / variabile locale | registro |
|------------------------------|----------|
| int i                        | \$a0     |
| int val                      | \$a1     |

| Parametro / variabile locale | registro |
|------------------------------|----------|
| int *p                       | \$s0     |

#### 4. Codice MIPS di binary

```

BINARY:
    addi $sp, $sp, -8
    sw $ra, 4($sp)
    sw $s0, 0($sp)
    la $t0, LIST
    sll $t1, $a0, 2
    addu $s0, $t0, $t1
    bge $a0, $0, ELSEIF
    la $v0, LIST
    j ENDIF

ELSEIF:
    lw $t0, ($s0)
    bne $t0, $a1, ELSE
    move $v0, $s0
    j ENDIF

ELSE:
    srl $t0, $a0, 1
    subi $a0, $t0, 1
    addi $a1, $a1, 1
    jal BINARY

ENDIF:
    lw $s0, 0($sp)
    lw $ra, 4($sp)
    add $sp, $sp, 8
    jr $ra

```

#### “eserciziAssembler1” esercizio 10

Tradurre da C a MIPS il programma riportato. Il modello di memoria è quello standard MIPS e gli interi sono a 32 bit. Non si eseguono ottimizzazioni. Si facciano le seguenti ipotesi:

- non si usa il frame pointer
- le variabili locali sono allocate nei registri (se possibile)
- vanno salvati solo i registri necessari

Si svolgano i seguenti 3 punti:

1. Si descriva il segmento di dati statici, dando gli spiazziamenti rispetto ai due global pointer nelle due ipotesi indicate specificando se gli spiazziamenti sono positivi o negativi e si traducano in MIPS le dichiarazioni delle variabili globali
2. Si descrivano l'area di attivazione della funzione `fill` e l'allocazione delle variabili locali di `fill` nei registri
3. Si traduca in linguaggio macchina dell'intera funzione `fill`

Sullo stesso programma C, si usi adesso il frame pointer e si svolgano i seguenti due punti:

1. Si descriva l'area di attivazione della funzione `fill`
2. Si traduca in linguaggio macchina l'istruzione: `pnt = &rnd;` di `fill`

```

#define N 4
int idx = 0;
char str[N];

char init(int seed);

void fill(int len) {

```

```

char *pnt, rnd;
rnd = init(0) + init(1);
pnt = &rnd;
while (idx < len)
    str[idx++] = *pnt;
}

int main(void) {
    fill(N);
}

```

### 1. Segmento dati statici

| Contenuto simbolico | Offset rispetto a gp = 0x1000 8000 | Segno    |
|---------------------|------------------------------------|----------|
| str[3]              | 0x80007                            | Negativo |
| str[2]              | 0x80006                            | Negativo |
| str[1]              | 0x80005                            | Negativo |
| str[0]              | 0x80004                            | Negativo |
| idx                 | 0x80000                            | Negativo |

| Contenuto simbolico | Offset rispetto a gp = 0x1000 0000 | Segno    |
|---------------------|------------------------------------|----------|
| str[3]              | 0x7                                | Positivo |
| str[2]              | 0x6                                | Positivo |
| str[1]              | 0x5                                | Positivo |
| str[0]              | 0x4                                | Positivo |
| idx                 | 0x0                                | Positivo |

```

.data
.equ N, 4
IDX: .word 0
STR: .space 4

```

### 2. Area e registri di fill

| Contenuto simbolico | Offset rispetto a \$sp |
|---------------------|------------------------|
| \$ra                | 5                      |
| \$s0                | 1                      |
| rnd                 | 0                      |
| \$a0                | -                      |
| \$v0                | -                      |

| Parametro / variabile locale | Registro |
|------------------------------|----------|
| len                          | \$a0     |
| pnt                          | \$s0     |

### 3. Codice MIPS di fill

```

FILL:
addi $sp, $sp, -9 # $ra, $s0, rnd
sw $ra, 5($sp)    # $ra
sw $s0, 1($sp)    # $s0
addi $sp, $sp, -7 # $a0, $v0 !! allineamento !!
sw $a0, 0($sp)
li $a0, 0

```

```

jal INIT
addi $sp, $sp, -4
sw $v0, 0($sp)
li $a0, 1
jal INIT
move $t0, $v0
lw $t1, 0($sp)
addi $sp, $sp, 4
lw $a0, 0($sp)
addi $sp, $sp, 7
add $t0, $t0, $t1
sb $t0, 0($sp)
move $s0, $sp

WHILE:
lw $t1, IDX
bge $t1, $a0, END
la $t0, STR
addu $t0, $t0, $t1
lb $t1, 0($s0)
sb $t1, 0(t0)
lw $t0, IDX
addi $t0, $t0, IDX
sw $t0, IDX
j WHILE

END:
lw $s0, 1($sp)
lw $ra, 5($sp)
addi $sp, $sp, 9
jr $ra

```

#### 1. Area di attivazione di fill

| Contenuto simbolico | Offset rispetto a \$fp |
|---------------------|------------------------|
| \$fp                | 0                      |
| \$ra                | -4                     |
| \$s0                | -8                     |
| rnd                 | -9                     |
| \$a0                | -                      |
| \$v0                | -                      |

#### 2. Istruzione pnt = &rnd usando \$fp

```
addiu $s0, $fp, -9
```

**07/10/20**

### “eserciziAssembler2” esercizio 3

Tradurre da C a MIPS il programma riportato. Il modello di memoria è quello standard MIPS e gli interi sono a 32 bit. Non si eseguano ottimizzazioni. Si facciano le seguenti ipotesi:

- si usa il frame pointer
- le variabili locali sono allocate nei registri (se possibile)
- vanno salvati solo i registri necessari

Si svolgano i seguenti punti:

1. Si descriva il segmento dei dati statici, dando anche spiazamenti delle variabili rispetto a \$gp



2. Si traduca in MIPS la funzione `main`
3. Si descriva l'area di attivazione della funzione `fibonacci`, indicando l'indirizzo a cui puntano `$fp` e `$sp`
4. Si traduca in MIPS la funzione `fibonacci`
5. Si descrivano la stack e i registri usati prima della chiamata a `fibonacci(n-2)`. Si assuma che le chiamate vengano eseguite nell'ordine di scrittura.

```
int valore = 6;

int fibonacci(int n) {
    if (n <= 1)
        return n;
    else
        return fibonacci(n - 1) + fibonacci(n - 2);
}

void main(void) {
    fibonacci(valore);
    return 0;
}
```

1. Segmento dati globale:

```
.data
VALORE: .word 6
```

| contenuto simbolico | indirizzo   | offset      |
|---------------------|-------------|-------------|
| VALORE              | 0x1000 0000 | 0x0000 8000 |

2. Funzione `main`;

```
.text
MAIN:
    lw $a0, VALORE
    jal FIBONACCI
    li $v0, 10
    syscall
```

3. Area di attivazione di `fibonacci`

| contenuto simbolico | offset (\$sp) | offset (\$fp) |
|---------------------|---------------|---------------|
| \$fp                | -4            | 0             |
| \$ra                | 0             | -4            |

4. Funzione `fibonacci`:

```
FIBONACCI:
    addi $sp, $sp, -8
    sw $fp, 4($sp)
    sw $ra, 0($sp)
    addiu $fp, $sp, 4
    bgt $a0, 1, ELSE
    move $v0, $a0
    j END

ELSE:
    addi $sp, $sp, -4
    sw $a0, -8(fp)
    subi $a0, $a0, 1
    jal FIBONACCI
```

```

addi $sp, $sp, -4
lw $a0, -8($fp)
sw v0, -8($fp)
subi $a0, $a0, 2
jal FIBONACCI
lw $t0, -8($fp)
add $v0, $v0, $t0
addiu $sp, $sp, 4

```

END:

```

lw $ra, 4($fp)
lw $fp, 0($fp)
addiu $sp, $sp, 8
jr $ra

```

5. Contenuto stack:

| contenuto simbolico | indirizzo   | valore |
|---------------------|-------------|--------|
| ...                 | 0x7FFF FFFC | ???    |
| \$fp                | 0x7FFF FFF8 | ???    |
| \$ra                | 0x7FFF FFF4 | ???    |
| \$v0                | 0X7FFF FFF0 | ???    |

Registri:

| registro | contenuto (hex) | significato      |
|----------|-----------------|------------------|
| \$fp     | 0x7FFF FFF8     | frame pointer    |
| \$sp     | 0x7FFF FFF0     | stack pointer    |
| \$a0     | 0x0000 0004     | n - 2            |
| \$v0     | 0x0000 0005     | fibonacci(n - 1) |

### “LogicaCombinatoria” esercizio 11

Si vuole progettare una rete combinatoria che riceve in ingresso 3 bit (A, B, C) e fornisce un'uscita U. Il risultato deve essere 1 se il numero di 1 negli ingressi è dispari, 0 altrimenti.

1. Completare la tabella di verità
2. Progettare la rete in prima forma canonica (SoP)
3. Scrivere una equazione equivalente a quella trovata in 2 contenente solamente operatori XOR

1. Tabella di verità:

| A | B | C | U |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

2. SoP:

$$U = \bar{A}\bar{B}C + \bar{A}C\bar{B} + A\bar{B}\bar{C} + ABC$$

3. Riduzione:

$$\begin{aligned}
 U &= \bar{A}BC + \bar{A}C\bar{C} + A\bar{B}C + ABC = \\
 &= \bar{A}(\bar{B}C + B\bar{C}) + A(\bar{B}C + BC) = \\
 &= \bar{A}(B \oplus C) + A(B \oplus C) = \\
 &= A \oplus B \oplus C
 \end{aligned}$$

### “LogicaSequenziale” esercizio 3

Sia dato il circuito sequenziale descritto dalle equazioni logiche:

$$\begin{aligned}
 D1 &= IN(Q1Q2 + Q1\bar{Q}2) + \bar{I}N(\bar{Q}1Q2 + Q1\bar{Q}2) = \\
 &= IN \oplus Q1 \oplus Q2 \\
 D2 &= \bar{Q}1 \\
 Z &= Q1Q2
 \end{aligned}$$

Il circuito è composto da due bistabili di tipo flip-flop master-slave di tipo D ed è dotato di ingresso  $IN$  e di uscita  $Z$ .

1. Disegnare lo schema del circuito
2. Completare il diagramma temporale
  - Si trascurino ritardi di propagazione
  - La commutazione avviene sul fronte di discesa del clock
  - $IN$  può variare in ogni momento

1. Circuito: (vedi figura)

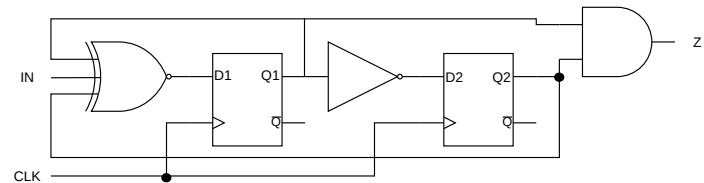


Figure 1: Circuito es 3

2. Diagramma temporale: (vedi figura)

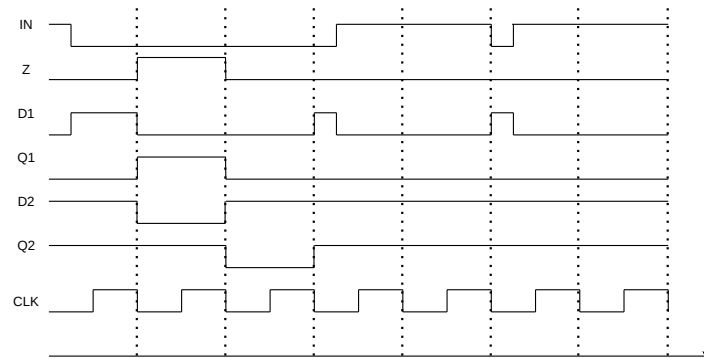


Figure 2: Diagramma temporale es 3

### “LogicaSequenziale” esercizio 4

Sia dato il seguente circuito sequenziale: (vedi figura)

Il circuito è composto da due bistabili di tipo flip-flop master-slave di tipo D ed è dotato di ingressi  $IN1, IN2, SEL$  e di uscita  $U$ . Completare il diagramma temporale. Si considerino le seguenti ipotesi:

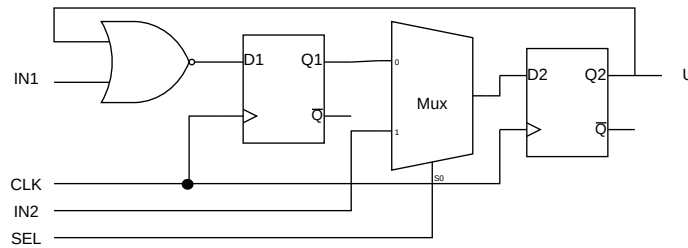


Figure 3: Circuito es 4

- Si trascurino ritardi di propagazione
- La commutazione avviene sul fronte di discesa del clock
- IN può variare in ogni momento

Soluzione in figura.

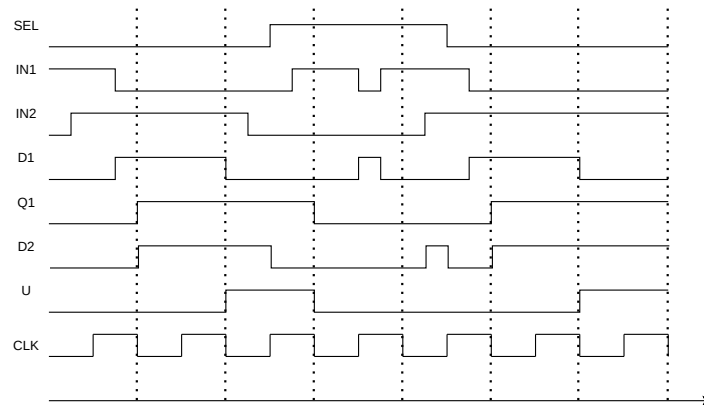


Figure 4: Diagramma temporale es 4

### “LogicaSequenziale” esercizio 7

Sia dato il circuito sequenziale descritto dalle equazioni logiche:

$$D = I \oplus Q$$

$$Z1 = \bar{Q}$$

$$Z2 = QD$$

Il circuito è composto da un bistabile di tipo flip-flop master-slave di tipo D ed è dotato di ingresso  $I$  e di uscite  $Z1, Z2$ .

1. Disegnare lo schema del circuito
2. Completare il diagramma temporale
  - Si trascurino ritardi di propagazione
  - La commutazione avviene sul fronte di discesa del clock
  - IN può variare in ogni momento

1. Circuito: (vedi figura)
2. Diagramma temporale: (vedi figura)

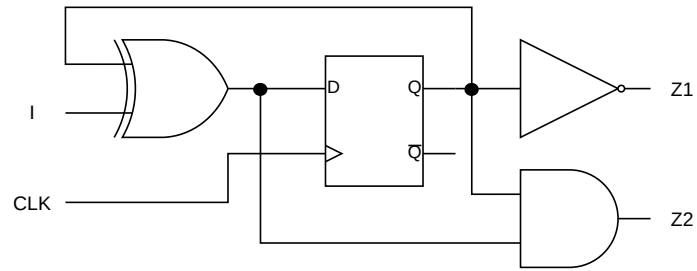


Figure 5: Circuito es 7

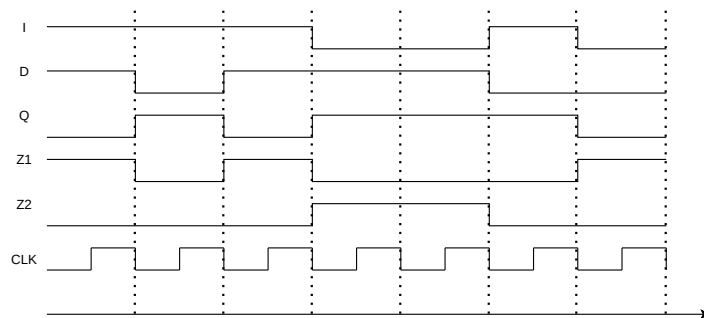


Figure 6: Diagramma temporale es 7