

## Il progetto di sistemi a livello RT

cap. 13 Introduction to Digital Systems - Wiley



Milos Ercegovac, Tomas Lang, Jaime Moreno  
*University of California*

## I sistemi a livello RT

- grafi di esecuzione
  - classificazione in funzione del sequenziamento
- organizzazione dei sistemi:
  - unità funzionale, unità di controllo
- organizzazione dei sistemi a livello RT:
  - sottosistemi dati e controllo
- specifica in VHDL di sistemi a livello RT
- analisi e progetto di sistemi a livello RT

03\_Il progetto di sistemi a livello RT

## Il livello RT

Caratterizzato da:

- Esistenza di sottosistema dati (datapath) e sottosistema controllo
- Stato del sottosistema dati: contenuto di un insieme di registri
- Funzione: sequenza di trasferimenti tra registri (1 o più cicli di clock)

03\_Il progetto di sistemi a livello RT

Trasferimento tra registri:

- Trasformazione sui dati mentre sono trasferiti da un registro ad un altro
- Sequenza di trasferimento controllata dal sottosistema di controllo.

03\_Il progetto di sistemi a livello RT

## Esempio: valutazione di polinomio

$$P_7(x) = \sum_{i=0}^7 p_i x^i$$

Evitare il calcolo delle potenze, uniche operazioni addizioni e moltiplicazioni:

- I soluzione: polinomi di Horn

$$P_7(x) = ((((((p_7x + p_6)x + p_5)x + p_4)x + p_3)x + p_2)x + p_1)x + p_0)$$

- II soluzione:

$$P_7(x) = (x^2)(x^2)[x^2(p_7x + p_6) + (p_5x + p_4)] + x^2(p_3x + p_2) + (p_1x + p_0)$$

03\_Il progetto di sistemi a livello RT

## Grafici di esecuzione

Grafo orientato:

- Nodi: operazioni
- Archi: precedenze tra operazioni
- Un nodo viene eseguito quando sono soddisfatte tutte le precedenze
  - AND: tutte le precedenze soddisfatte
  - OR: almeno una precedenza soddisfatta

Grafo sequenziale:

- 1 solo nodo attivo alla volta

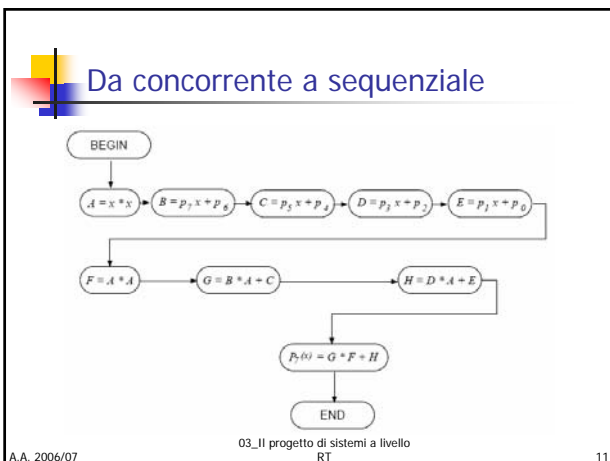
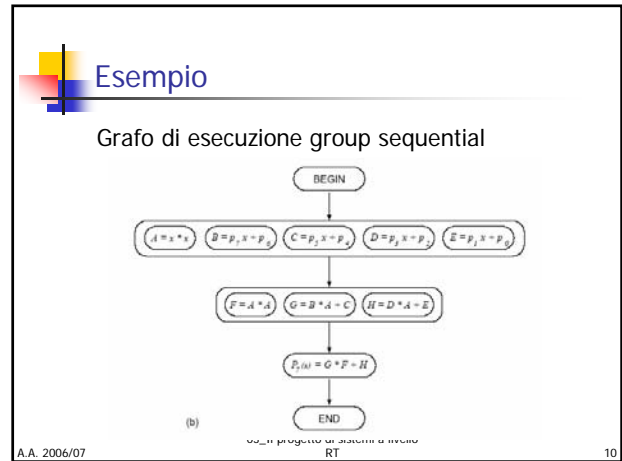
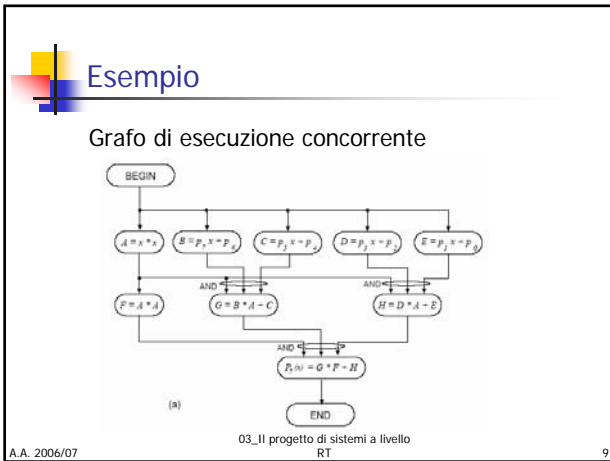
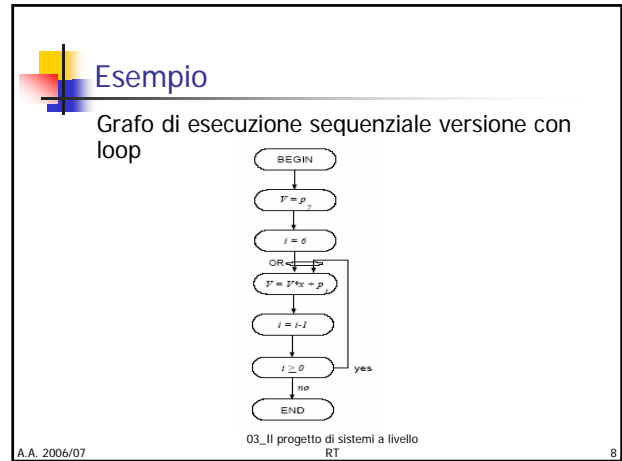
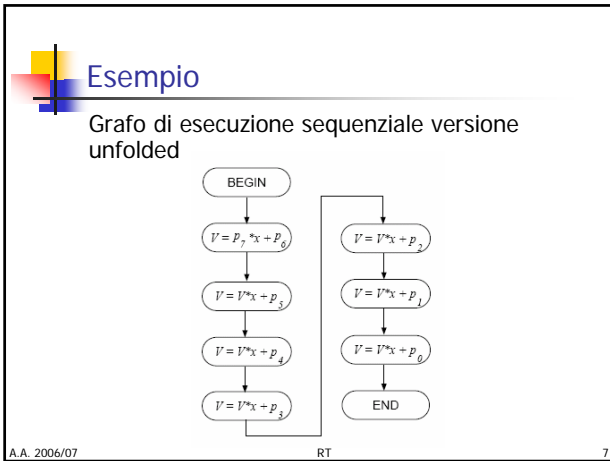
Grafo concorrente:

- più nodi attivi alla volta.


Grafo group sequential:

- sequenzialità tra gruppi, concorrenza all'interno di un gruppo.

03\_Il progetto di sistemi a livello RT



- ### Vantaggi dei grafi sequenziali
- semplicità di sviluppo: facile tener traccia delle trasformazioni di dato
  - stato determinato da singola operazione o gruppo di operazioni
  - controllo di sequenzializzazione semplice:
    - un solo arco da un nodo all'unico successore
    - più archi uscenti, ma uno solo attivo condizionalmente
- A.A. 2006/07 03\_II progetto di sistemi a livello RT 12





---

- Sequenzializzazione: FSM in cui ogni nodo o gruppo è uno stato del controllore

Vantaggi dei grafi concorrenti:  
implementazioni potenzialmente più veloci

A.A. 2006/07 03\_II progetto di sistemi a livello RT 13




## Organizzazione dei sistemi

---

Due funzioni:

- trasformazione dei dati → unità funzionali
- controllo delle trasformazioni e sequenzializzazione → unità di controllo

A.A. 2006/07 03\_II progetto di sistemi a livello RT 14




## Classificazione rispetto alle unità funzionali

---

- sistemi non condivisi
- sistemi condivisi
- sistemi mono-modulo

A.A. 2006/07 03\_II progetto di sistemi a livello RT 15



## Sistemi non condivisi


---

Corrispondenza 1:1 tra nodi del grafo di esecuzione e unità funzionali.

Connessioni tra unità funzionali: archi del grafo di esecuzione

Vantaggi: semplicità  
Svantaggi: molti moduli, magari non usati efficientemente, interconnessioni tra moduli complesse

A.A. 2006/07 03\_II progetto di sistemi a livello RT 16




## Sistemi condivisi

---

Unità funzionali che eseguono operazioni in più nodi del grafo in tempi diversi.

Vantaggi: condivisione delle risorse

A.A. 2006/07 03\_II progetto di sistemi a livello RT 17



## Sistemi mono-modulo

---

Caso estremo di condivisione: 1 solo modulo esegue le operazioni di tutti i nodi del grafo di esecuzione.

A.A. 2006/07 03\_II progetto di sistemi a livello RT 18

### Classificazione rispetto al controllo

- controllo centralizzato
- controllo decentralizzato
- controllo semicentralizzato

A.A. 2006/07 03\_II progetto di sistemi a livello RT 19

### Controllo centralizzato

Unica unità di controllo per tutto il grafo di esecuzione.  
 Genera segnali di controllo e riceve segnali di condizione.

A.A. 2006/07 03\_II progetto di sistemi a livello RT 20

### Controllo decentralizzato

Ogni modulo ha il suo controllo.  
 Coordinamento del sistema: connessioni tra i moduli.

A.A. 2006/07 03\_II progetto di sistemi a livello RT 21

### Controllo semicentralizzato

Ogni modulo ha il suo controllo.  
 Coordinamento del sistema: esiste un unico sequenzializzatore.

A.A. 2006/07 03\_II progetto di sistemi a livello RT 22

### Esempio

Valutatore di polinomi di grado 7, grafo delle esecuzioni concorrente: sistema non condiviso, controllo decentralizzato.

Ogni box è un modulo con:

- operatore (sommatore, moltiplicatore)
- memorizzazione di operandi e risultati
- controllo locale dell'esecuzione.

Al massimo 5 moduli attivi simultaneamente.

A.A. 2006/07 03\_II progetto di sistemi a livello RT 23

A.A. 2006/07 03\_II progetto di sistemi a livello RT 24

Riduzione dei moduli a parità di tempo di esecuzione:

- modulo più complesso

if  $c1 = 1$  then  $O_1 = I_{11} \times I_{12} + I_{13}$   
 if  $c2 = 1$  then  $O_2 = I_{21} \times I_{22} + I_{23}$

A.A. 2006/07 03\_II progetto di sistemi a livello RT 25

Controllo centralizzato, grafo di esecuzione group sequential.

Step1:  $A \rightarrow x^*y$ ,  $B \rightarrow p_1 x^* p_0$ ,  $C \rightarrow p_2 x^* p_1$ ,  $D \rightarrow p_3 x^* p_2$ ,  $E \rightarrow p_4 x^* p_3$

Step2:  $F \rightarrow A^*A$ ,  $G \rightarrow B^*A + C$ ,  $H \rightarrow D^*A + E$

Step3:  $F_7(x) = G^*F + H$

A.A. 2006/07 03\_II progetto di sistemi a livello RT 26

CONTROL

Step 3  
Step 2  
Step 1

Computation steps:  
 Step 1: A, B, C, D, E  
 Step 2: F, G, H  
 Step 3: F(x)

A.A. 2006/07 03\_II progetto di sistemi a livello RT 27

Esempio

Valutatore di polinomi di grado 7, grafo delle esecuzioni sequenziale (con ciclo): sistema mono-modulo, controllo centralizzato.

Modulo M:  $z \leftarrow a \times b + c$

Register file R: lettura contemporanea di 2 valori, scrittura di uno.

A.A. 2006/07 03\_II progetto di sistemi a livello RT 28

Algoritmo: ciclo:

- trasferimento dei dati dal register file ai registri a, b, c
- calcolo di z
- salvataggio del risultato

- 1:  $a \leftarrow p_7, b \leftarrow x$
- 2:  $c \leftarrow p_6$
- 3:  $z \leftarrow p_7 x + p_6$
- 4:  $T \leftarrow z$

A.A. 2006/07 03\_II progetto di sistemi a livello RT 29

CONTROL

REGISTER ARRAY R

DISTRIBUTOR

OPERATOR  $a * b + c$

register

A.A. 2006/07 RT 30

### Sistemi a livello RT

Grafi di esecuzione: sequenziali o group sequential  
 Controllo: centralizzato  
 Sottosistema dati & sottosistema controllo.

A.A. 2006/07 31

### Specifica VHDL di sistemi a livello RT

Specifica: calcolare  
 INPUTS:  $x, y \in \{-128, \dots, 127\}$   
 OUTPUT:  $z \in \{-256, \dots, 508\}$

FUNCTION:  $z = \begin{cases} 4[(x + |y|)/2] & \text{if } x < |y| \\ 4x & \text{otherwise} \end{cases}$

senza usare sommatori a due operandi.

A.A. 2006/07 32

$$a = (x + |y|)/2 = x + (|y| - x)/2$$

- inizializzare a a x
- incrementare a e decrementare |y| mentre  $a < |y|$

A.A. 2006/07 33

A.A. 2006/07 34

### Package/entity

```

PACKAGE inc_dec_pkg IS
    SUBTYPE SignDataT IS INTEGER RANGE -128 TO 127;
    SUBTYPE PosDataT IS INTEGER RANGE 0 TO 128;
    SUBTYPE DataoutT IS INTEGER RANGE -256 TO 508;
END inc_dec_pkg;

USE WORK.inc_dec_pkg.ALL;
ENTITY inc_dec IS
    PORT(x_in,y_in: IN SignDataT;
         z_out : OUT DataoutT;
         clk : IN BIT);
END inc_dec;
    
```

A.A. 2006/07 35

### Architecture

```

ARCHITECTURE high_level OF inc_dec IS
BEGIN
    PROCESS (clk)
        VARIABLE a : SignDataT;
        VARIABLE w : PosDataT;
    BEGIN
        IF (clk'EVENT AND clk = '1') THEN
            a:= x_in;
            w:= ABS(y_in);
            WHILE (a < w) LOOP
                a:= a+1;
                w:= w-1;
            END LOOP;
            z_out <= (a * 4);
        END IF;
    END PROCESS;
END high_level;
    
```

A.A. 2006/07 36

### Metodologia di specifica

- Disegnare il grafo delle esecuzioni
- Scrivere in VHDL la descrizione del sistema sulla base del grafo.

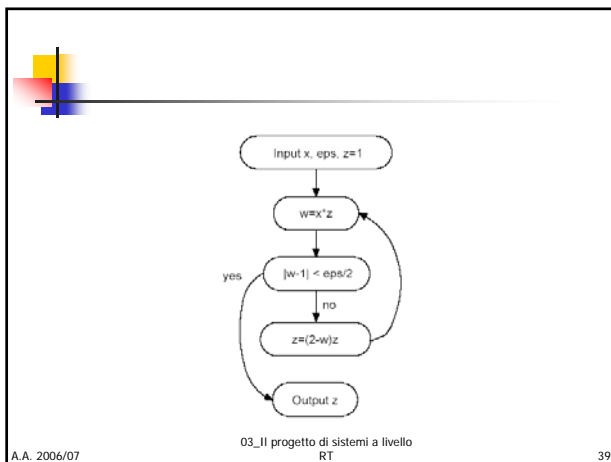
A.A. 2006/07 03\_II progetto di sistemi a livello RT 37

### Esempio

Ricorrenza di Newton-Raphson per calcolare un'approssimazione  $z$  di  $1/x$ , dove  $1/2 \leq x \leq 1$ .  
 Errore di approssimazione  $\epsilon$ .  
 $z_0 = 1$   
 $z_{i+1} = z_i (2 - x z_i)$

Terminazione:  $x z_k - 1 < 1/2 \epsilon$  che equivale a  $z_k - 1/x < \epsilon$  essendo  $x \geq 1/2$

A.A. 2006/07 03\_II progetto di sistemi a livello RT 38



### Package/entity

```

PACKAGE recip_pkg IS
  SUBTYPE DatainT IS REAL RANGE 0.5 TO 1.0;
  SUBTYPE DataoutT IS REAL RANGE 1.0 TO 2.0;
  SUBTYPE EpsT IS REAL RANGE 0.0 TO 0.1;
END recip_pkg;

USE WORK.recip_pkg.ALL;
ENTITY reciprocal IS
  PORT(x_in : IN DatainT ;
        eps_in : IN EpsT ;
        z_out : OUT DataoutT;
        clk : IN BIT );
END reciprocal;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 40

### Architecture

```

ARCHITECTURE high_level OF reciprocal IS
BEGIN
  PROCESS (clk)
    VARIABLE x,w: DatainT ;
    VARIABLE z : DataoutT;
  BEGIN
    IF (clk EVENT AND clk = '1') THEN
      z := 1.0; x := x_in;
      w := x * z;
      WHILE (ABS(w - 1.0) > eps_in/2.0) LOOP
        z := (2.0 - w) * z;
        w := x * z;
      END LOOP;
      z_out <= z;
    END IF;
  END PROCESS;
END high_level;
    
```

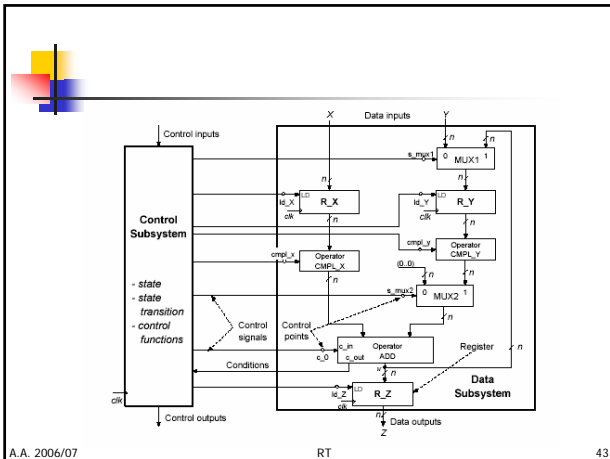
A.A. 2006/07 03\_II progetto di sistemi a livello RT 41

### Implementazione di sistemi a livello RT

- Sottosistema dati:
  - segnali di dato in ingresso
  - segnali di dato in uscita
  - valori intermedi
  - condizioni sui dati
- Sottosistema controllo:
  - segnali di controllo in ingresso
  - segnali di controllo in uscita
  - segnali di controllo per il sottosistema dati.

Tipi: bit, bit-vector, array di bit-vector.

A.A. 2006/07 03\_II progetto di sistemi a livello RT 42



A.A. 2006/07 RT 43

- Operazioni sincronizzate da un clock
  - In ogni ciclo di clock avviene uno o più trasferimenti tra registri
    - sorgente: registro o uscita di operatore alimentato da registri
    - destinazione: registro
- $R_Z := ADD(R_x, R_Y, 1)$

A.A. 2006/07 03\_II progetto di sistemi a livello RT 44

- Trasferimento: segnali di controllo che stabiliscono la connessione: per eseguire  $R_Z := ADD(R_X, R_Y, 1)$  bisogna assegnare come segue i segnali di controllo:
  - $cmpl\_x=0, cmpl\_y=0, s\_mux2=1, c\_0=1, ld\_Z=1$
- operatori implementabili in hardware (ADD e CMPL)

A.A. 2006/07 03\_II progetto di sistemi a livello RT 45

- RT-group: gruppo di trasferimenti fra registri contemporanei in un ciclo di clock, assenza di conflitti.
- Esempio di conflitto:  
 $R_Z := ADD(R_X, R_Y, 0)$  e  
 $R_Y := ADD(R_Y, 0, 0)$
- Il grafo di esecuzione è implementato da una sequenza di RT groups:  
 RT-group1:  $R_X=X; R_Y=Y$   
 segnali:  $ld\_X=1, ld\_Y=1$

A.A. 2006/07 03\_II progetto di sistemi a livello RT 46

- RT-group2:  $R_X=X; R_Z= R_Y - R_X$   
 segnali:  $ld\_X=1, ld\_Z=1, smux2=1, cmpl\_x=1, c\_0=1$
- RT-group3:  $R_Y=R_X - R_Y; R_Z = R_X - R_Y$   
 segnali  $ld\_Y=1, ld\_Z=1, smux1=1, cmpl\_y=1, smux2=1, c\_0=1$

A.A. 2006/07 03\_II progetto di sistemi a livello RT 47

Entity

```

ENTITY data_sub IS
  PORT(X, Y      : IN BIT_VECTOR ;
        ld_X, ld_Y, ld_Z : IN BIT ;
        s_mux1, s_mux2 : IN BIT ;
        cmpl_x, cmpl_y : IN BIT ;
        c_0, clk      : IN BIT ;
        Z             : OUT BIT_VECTOR;
        c_out        : BIT);
END data_sub;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 48

### Architecture

```

ARCHITECTURE behavioral OF data_sub IS
  SIGNAL R_X,R_Y,R_Z: BIT_VECTOR(n-1 DOWNT0 0);
  BEGIN
  PROCESS (clk)
    VARIABLE zero_n: BIT_VECTOR(n-1 DOWNT0 0):= (OTHERS => '0');
    VARIABLE w : BIT_VECTOR(n-1 DOWNT0 0);
    VARIABLE selec : BIT_VECTOR( 2 DOWNT0 0);
  BEGIN
    selec:= cml_x & cml_y & smux_2;
  CASE selec IS
    WHEN "000" => w := add(R_X,zero_n,c_0);
    WHEN "001" => w := add(R_X,R_Y,c_0) ;
    WHEN "010" => w := add(R_X,zero_n,c_0);
    WHEN "011" => w := add(R_X,cml(R_Y),c_0) ;
    WHEN "100" => w := add(cml(R_X),zero_n,c_0);
    WHEN "101" => w := add(cml(R_X),R_Y,c_0) ;
    WHEN "110" => w := add(cml(R_X),zero_n,c_0);
    WHEN "111" => w := add(cml(R_X),cml(R_Y),c_0);
  END CASE;
  END PROCESS;
  END behavioral;
  
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 49

```

IF (clk'EVENT AND clk = '1') THEN
  IF (ld_X = '1') THEN R_X <= X; END IF;
  IF (ld_Z = '1') THEN R_Z <= w; END IF;
  IF (ld_Y = '1') THEN
    IF (smux_1 = '0') THEN R_Y <= Y;
    ELSE R_Y <= w;
    END IF;
  END IF;
END IF;
END PROCESS;
END behavioral;
  
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 50

### Progetto di sistemi a livello RT

Partenza: grafo di esecuzione corrispondente a specifica comportamentale VHDL.

1- Progetto del sottosistema dati:

- determinare gli operatori (unità funzionali):  
**due operazioni possono essere assegnate alla stessa unità funzionale se fanno parte di RT-group diversi**

A.A. 2006/07 03\_II progetto di sistemi a livello RT 51

- determinare i registri necessari per gli operandi, i risultati e i risultati intermedi:  
**due variabili assegnabili allo stesso registro se attive in tempi disgiunti**
- connettere i componenti con fili e multiplexer come richiesto dai trasferimenti nella sequenza
- determinare i segnali di controllo e le condizioni richieste dalla sequenza
- descrivere la struttura del sottosistema dati come descrizione strutturale VHDL

A.A. 2006/07 03\_II progetto di sistemi a livello RT 52

- descrivere in VHDL il comportamento della sottostruttura dati come trasferimenti che avvengono per ogni segnale di controllo.

```

ARCHITECTURE generalized OF data_subsystem IS
  BEGIN
  PROCESS (clk)
    SIGNAL reg_A,reg_B: BIT_VECTOR;
  BEGIN
    IF (clk'EVENT AND clk = '1') THEN
      IF (ctrl0 = '1') THEN ... END IF;
      IF (ctrl1 = '1') THEN ... END IF;
      ....
      IF (ctrlj = '1') THEN ... END IF;
    END IF;
  END PROCESS;
  END generalized;
  
```


A.A. 2006/07 03\_II progetto di sistemi a livello RT 53

2- Descrivere strutturalmente in VHDL l'interfaccia tra sottosistema dati e controllo con segnali di controllo e di condizione.

```

ENTITY group_seq_system IS
  PORT (data_in : IN BIT_VECTOR;
        data_out: OUT BIT_VECTOR;
        ctrl_in : IN BIT_VECTOR;
        ctrl_out: OUT BIT_VECTOR;
        clk : IN BIT);
END group_seq_system;
  
```


A.A. 2006/07 03\_II progetto di sistemi a livello RT 54



```

ARCHITECTURE generalized OF group_seq_system IS
SIGNAL controls : BIT_VECTOR;
SIGNAL conds : BIT_VECTOR;
BEGIN
  U1: ENTITY data_subsystem
      PORT MAP (data_in,data_out,controls,conds,clk);
  U2: ENTITY control_subsystem
      PORT MAP (ctrl_in,ctrl_out,conds,controls,clk);
END generalized;
    
```


A.A. 2006/07 03\_II progetto di sistemi a livello RT 55



3- Progetto del sottosistema controllo:

- determinare la sequenza di trasferimenti RT per i dati
- assegnare uno stato a ogni RT-group, determinare e descrivere in VHDL le funzioni di transizione di stato e di uscita
- implementare la FSM.


A.A. 2006/07 03\_II progetto di sistemi a livello RT 56



```

ARCHITECTURE generalized OF control_subsystem IS
BEGIN
  PROCESS (clk)
    TYPE stateT IS (s0, s1, sk);
    SIGNAL state: stateT:= s0;
  BEGIN
    IF (clk'EVENT AND clk = '1') THEN
      CASE state IS
        WHEN s0 => ...;
        WHEN s1 => ...;
        WHEN sk => ...;
      END CASE;
    END IF;
  END PROCESS;
END generalized;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 57




## Progetto di moltiplicatore

Moltiplicatore di interi positivi  $x$  e  $y$  nell'intervallo  $0 \dots 2^n - 1$ , con risultato nell'intervallo  $0 \dots (2^{2n} - 2^{n+1} + 1)$

INPUTS:  $x, y \in \{0, 1, \dots, 2^n - 1\}$   
 OUTPUT:  $z \in \{0, 1, \dots, 2^{2n} - 2^{n+1} + 1\}$   
 FUNCTION:  $z = x \times y$

A.A. 2006/07 03\_II progetto di sistemi a livello RT 58




## Algoritmo: serie-parallelo

Bit-vector per  $x = (x_{n-1} \dots x_0)$ ,  $y = (y_{n-1} \dots y_0)$  e  $z = (z_{2n-1} \dots z_0)$

$n$  iterazioni della ricorrenza:  
 $z[i+1] = (1/2)(z[i] + (x \cdot 2^n) \cdot y_i)$   
 $z[0] = 0$

risultato  $z = z[n]$

A.A. 2006/07 03\_II progetto di sistemi a livello RT 59



- passo 1: moltiplicazione di  $x$  per  $2^n$
- passo 2: risultato precedente moltiplicato per  $y_i$
- passo 3: risultato precedente sommato a  $z[i]$
- risultato precedente diviso per 2 e memorizzato come  $z[i+1]$

A.A. 2006/07 03\_II progetto di sistemi a livello RT 60

### Esempio

$x = 1011 (11_{10})$        $y = 0101 (5_{10})$   
 $n = 4$   
 $z = x \cdot y = 00110111 = (55_{10})$

$z[0] = 0000\ 0000$   
 $x \cdot 2^n = 1011\ 0000$

A.A. 2006/07      03\_II progetto di sistemi a livello RT      61

**passo 1:**  
 $(x \cdot 2^n) \cdot y_0 = 1011\ 0000$   
 $z[0] + (x \cdot 2^n) \cdot y_0 = 1011\ 0000$   
 $z[1] = 1/2 (z[0] + (x \cdot 2^n) \cdot y_0) = 0101\ 1000$

**passo 2:**  
 $(x \cdot 2^n) \cdot y_1 = 0000\ 0000$   
 $z[1] + (x \cdot 2^n) \cdot y_1 = 0101\ 1000$   
 $z[2] = 1/2 (z[1] + (x \cdot 2^n) \cdot y_1) = 0010\ 1100$

A.A. 2006/07      03\_II progetto di sistemi a livello RT      62

**passo 3:**  
 $(x \cdot 2^n) \cdot y_2 = 1011\ 0000$   
 $z[2] + (x \cdot 2^n) \cdot y_2 = 1101\ 1100$   
 $z[3] = 1/2 (z[2] + (x \cdot 2^n) \cdot y_2) = 0110\ 1110$

**passo 4:**  
 $(x \cdot 2^n) \cdot y_3 = 0000\ 0000$   
 $z[3] + (x \cdot 2^n) \cdot y_3 = 0110\ 1110$   
 $z[4] = 1/2 (z[3] + (x \cdot 2^n) \cdot y_3) = 0011\ 0111$   
 $z = z[4] = 0011\ 0111$

A.A. 2006/07      03\_II progetto di sistemi a livello RT      63

- moltiplicazione di x per  $2^n$ : shift a sinistra di x n volte e inserimento di 0
- moltiplicazione per  $y_i$ : o 0 o  $x \cdot 2^n$
- somma: sommatore
- divisione per 2: shift a destra di 1 posizione.

Si usa un bit di y alla volta (seriale) e tutti i bit di x (parallelo)  $\Rightarrow$  algoritmo serie-parallelo.

A.A. 2006/07      03\_II progetto di sistemi a livello RT      64

### Grafo delle esecuzioni

```

graph TD
    idle((idle)) -- start --> input[Input x, y  
z=0, i=0]
    input --> loop[z = (1/2)(z + (x * 2^n) * y_i)  
i=i+1]
    loop -- i = n-1 --> idle
    loop -- i ≠ n-1 --> loop
    
```

A.A. 2006/07      03\_II progetto di sistemi a livello RT      65

### Specifica VHDL

```

ENTITY multiplier IS
  GENERIC(n : NATURAL:= 16);
  PORT (start : IN BIT ;
        xin,yin: IN UNSIGNED(n-1 DOWNT0 0);
        clk : IN BIT ;
        zout : OUT UNSIGNED(2*n-1 DOWNT0 0);
        done : OUT BIT);
END multiplier;
    
```

A.A. 2006/07      03\_II progetto di sistemi a livello RT      66

```

ARCHITECTURE behavioral OF multiplier IS
  TYPE stateT IS (idle,setup,active) ;
  SIGNAL state : stateT := idle ;
  SIGNAL x,y : BIT_VECTOR(n-1 DOWNTO 0);
  SIGNAL z : BIT_VECTOR(2*n-1 DOWNTO 0);
  SIGNAL count : NATURAL RANGE 0 TO n-1 ;
  BEGIN
    PROCESS (clk)
      VARIABLE zero_2n : UNSIGNED(2*n-1 DOWNTO 0);
      VARIABLE scale : UNSIGNED(n-1 DOWNTO 0) ;
      VARIABLE add_out : UNSIGNED(2*n DOWNTO 0) ;
      BEGIN
        zero_2n:= (OTHERS => '0');
        scale := (OTHERS => '0');
        IF (clk'EVENT AND clk = '1') THEN
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 67

```

      CASE state IS
        WHEN idle => done <= '1';
          IF (start = '1') THEN state <= setup;
          ELSE state <= idle;
          END IF;
        WHEN setup => x <= xin; y <= yin; z <= zero_2n;
          count <= 0; zout <= zero_2n; done <= '0';
          state <= active;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 68

```

        WHEN active =>
          IF (y(count)) = '0' THEN
            add_out := '0' & z;
          ELSE
            add_out := ('0' & z) + ('0' & x & scale);
          END IF
          z <= add_out(2*n DOWNTO 1);
          zout <= add_out(2*n DOWNTO 1);
          IF (count /= (n-1)) THEN
            state <= active;
            count <= count+1;
          ELSE
            state <= idle;
            done <= '1';
          END IF;
        END CASE;
      END IF;
    END PROCESS;
  END behavioral;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 69

### Implementazione del datapath

$z[i+1] = (1/2)(z[i] + (x \cdot 2^n) \cdot y_i)$  può essere implementata all'interno di un singolo gruppo del grafo

Componenti e connessioni:

- 1 registro a n-bit x per memorizzare x
- 1 registro a 2n-bit z per memorizzare z. Il registro z può essere messo a 0, serve un clear

A.A. 2006/07 03\_II progetto di sistemi a livello RT 70

- 1 registro a n-bit y per memorizzare y: shift register con caricamento parallelo e uscita seriale
- 1 modulo per calcolare  $x \cdot 2^n$ : shift a sinistra di n posizioni, connettere x con gli input del sommatore allineati con gli n bit più significativi di z
- 1 modulo per calcolare  $(x \cdot 2^n) \cdot y_i$ : n AND a 2 ingressi

A.A. 2006/07 03\_II progetto di sistemi a livello RT 71

- 1 modulo per la somma di n bit:  
 $(x \cdot 2^n) \cdot y_i$  xxxxxxxx 00000000  
 z[i]            zzzzzzzz zzzzzzzz
- 1 modulo per dividere per 2 e caricare il risultato in z: caricare z con la concatenazione di carry-out e somma del sommatore (n-1 bit più a sinistra)

A.A. 2006/07 03\_II progetto di sistemi a livello RT 72

Segnali di controllo:

ldX, ldY, ldZ: caricamento dei registri X, Y, Z

shY: shift a destra di Y di 1 posizione

clrZ: caricamento di 0 in Z

Non vengono generati segnali di condizione.

A.A. 2006/07 03\_II progetto di sistemi a livello RT 73

### Entity

```

ENTITY multdata_bhv IS
  GENERIC(n : NATURAL := 16);
  PORT (xin,yin : IN UNSIGNED(n-1 DOWNT0 0);
        ldX,ldY,ldZ : IN BIT;
        shY,clrZ : IN BIT;
        zout : OUT UNSIGNED(2*n-1 DOWNT0 0);
        clk : IN BIT);
END multdata_bhv;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 74

### Architecture (behavioral) del datapath

```

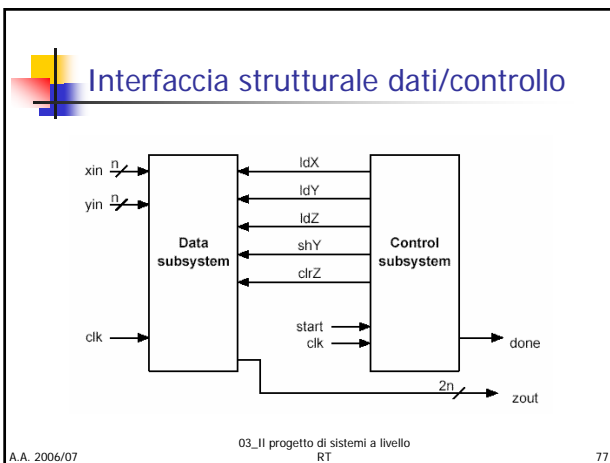
ARCHITECTURE behavioral OF multdata_bhv IS
  SIGNAL x,y : UNSIGNED(n-1 DOWNT0 0);
  SIGNAL z : UNSIGNED(2*n-1 DOWNT0 0);
BEGIN
  PROCESS(clk)
    VARIABLE zero_2n : UNSIGNED(2*n-1 DOWNT0 0);
    VARIABLE scale : UNSIGNED(n-1 DOWNT0 0);
    VARIABLE add_out : UNSIGNED(2*n DOWNT0 0);
  BEGIN
    zero_2n := (OTHERS => '0');
    scale := (OTHERS => '0');
  END PROCESS;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 75

```

IF (clk'EVENT AND clk = '1') THEN
  IF (ldX = '1') THEN x <= xin; END IF;
  IF (ldY = '1') THEN y <= yin; END IF;
  IF (y(0) = '0') THEN
    add_out := '0' & z;
  ELSE
    add_out := ('0' & z) + ('0' & x & scale);
  END IF;
  IF (ldZ = '1') THEN
    z <= add_out(2*n DOWNT0 1);
    zout <= add_out(2*n DOWNT0 1);
  END IF;
  IF (clrZ = '1') THEN z <= zero_2n; END IF;
  IF (shY = '1') THEN y <= '0' & y(n-1 DOWNT0 1); END IF;
END IF;
END PROCESS;
END behavioral;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 76



### Entity/Architecture

```

ENTITY multiplier IS
  GENERIC(n : NATURAL := 16);
  PORT (start : IN BIT;
        xin,yin : IN UNSIGNED(n-1 DOWNT0 0);
        clk : IN BIT;
        zout : OUT UNSIGNED(2*n-1 DOWNT0 0);
        done : OUT BIT);
END multiplier;

ARCHITECTURE structural OF multiplier IS
  SIGNAL ldX,ldY,ldZ,clrZ,shY : BIT;
BEGIN
  U1: ENTITY multdata_bhv
    PORT MAP (xin,yin,ldX,ldY,ldZ,shY,clrZ,zout,clk);
  U2: ENTITY multctrl
    PORT MAP (start,ldX,ldY,ldZ,shY,clrZ,done,clk);
END structural;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 78

### Datapath: descrizione strutturale

The diagram illustrates the structural description of the datapath. It features several key components:
 

- Register X:** Receives  $x_{in}$  and outputs  $x_{reg\_out}$ .
- Shift register Y:** Receives  $y_{in}$  and  $shY$ , and outputs  $y_0$ .
- AND gates VAND:** Take  $x_{reg\_out}$  and  $y_0$  as input to produce  $and\_out$ .
- Register Z:** Receives  $z_{reg\_in}$  and  $clrZ$ , and outputs  $z_{reg\_out}$ .
- Adder ADD:** Takes  $and\_out$  and  $z_{reg\_out}$  as input to produce  $add\_out$  and  $c\_out$ .

 The output  $zout$  is derived from the  $z_{reg\_out}$  and  $add\_out$  signals.

79

### Descrizione strutturale VHDL

```

ENTITY multdata_str IS
  GENERIC(n : NATURAL := 16);
  PORT (xin,yin : IN UNSIGNED(n-1 DOWNTO 0);
        ldX,ldY,ldZ : IN BIT;
        shY,clrZ : IN BIT;
        zout : OUT UNSIGNED(2*n-1 DOWNTO 0);
        clk : IN BIT);
END multdata_str;
  
```

80

### ARCHITECTURE structural OF multdata\_str IS

```

SIGNAL add_out, xreg_out, and_out : BIT_VECTOR(n-1 DOWNTO 0);
SIGNAL c_out, y_0 : BIT;
SIGNAL zreg_out : BIT_VECTOR(2*n-1 DOWNTO 0);
SIGNAL zreg_in : BIT_VECTOR(2*n-1 DOWNTO 0);
SIGNAL clr : BIT;
BEGIN
  zreg_in <= c_out & add_out & zreg_out(n-1 DOWNTO 1);
  X : ENTITY reg PORT MAP (xin, ldX, xreg_out, clk);
  Y : ENTITY shiftreg PORT MAP (yin, ldY, shY, y_0, clk);
  Z : ENTITY regclr PORT MAP (zreg_in, ldZ, clrZ, zreg_out, clk);
  VAND : ENTITY vectorand PORT MAP (xreg_out, y_0, and_out);
  ADD : ENTITY adder_pos PORT MAP (zreg_out(2*n-1 DOWNTO n),
    and_out,'0',add_out,c_out);
END structural;
  
```

81

### Implementazione unità di controllo

Diagramma degli stati derivato dal grafo di esecuzione:

The state transition diagram shows the following states and transitions:
 

- idle:** Initial state. Transitions to **setup** (start=1) and back to **idle** (done=1, controls=00000).
- setup:** Transitions to **active, count=0** (done=0, controls=11001).
- active, count=0, 1, ..., n-1:** A sequence of states where the count increases by 1 in each step. Transitions back to **idle** occur when the count reaches n-1 (controls=00110).

82

### Entity

```

ENTITY multctrl IS
  GENERIC(n : NATURAL := 16);
  PORT (start : IN BIT;
        ldX,ldY,ldZ : OUT BIT;
        shY, clrZ : OUT BIT;
        done : OUT BIT; clk : IN BIT);
END multctrl;
  
```


83

### Architecture

```

ARCHITECTURE behavioral OF multctrl IS
  TYPE stateT IS (idle,setup,active);
  SIGNAL state : stateT:= idle;
  SIGNAL count : NATURAL RANGE 0 TO n-1;
BEGIN
  PROCESS (clk)
  BEGIN
    IF (clk'EVENT AND clk = '1') THEN
      CASE state IS
        WHEN idle => IF (start = '1') THEN state <= setup;
          ELSE state <= idle ; END IF;
        WHEN setup => state <= active; count <= 0;
        WHEN active => IF (count = (n-1)) THEN count <= 0; state <= idle ;
          ELSE count <= count+1; state <= active; END IF;
      END CASE;
    END IF;
  END PROCESS;
  
```


84



```

PROCESS (state,count)
VARIABLE controls: BIT_VECTOR(5 DOWNT0 0); --ldX,ldY,ldZ,shY,clrZ)
BEGIN
CASE state IS
WHEN idle => controls := "100000";
WHEN setup => controls := "011001";
WHEN active => controls := "000110";
END CASE;
done <= controls(5);
ldX <= controls(4); ldY <= controls(3); ldZ <= controls(2);
shY <= controls(1); clrZ <= controls(0);
END PROCESS;
END behavioral;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 85




### Analisi di tempistica

Cycle time: frequenza del clock massima legata al cammino critico:

- $t_r$ : ritardo dei registri nel produrre uscite stabili, include ritardo di setup e di propagazione
- $t_{buf}$ : ritardo del buffer richiesto dal caricamento SU  $y_0$
- $t_{and}$ : ritardo delle porte AND
- $t_{add}$ : ritardo del sommatore

A.A. 2006/07 03\_II progetto di sistemi a livello RT 86



Tempo di esecuzione:  
 $n+2$  cicli

- 1 ciclo nello stato idle
- 1 ciclo nello stato setup
- $n$  cicli per le iterazioni.

A.A. 2006/07 03\_II progetto di sistemi a livello RT 87




### Sottosistemi dati e controllo

cap. 14 Introduction to Digital Systems - Wiley



Milos Ercegovac, Tomas Lang, Jaime Moreno  
 University of California



### Sottosistemi dati e controllo

- Componenti e organizzazione dei sottosistemi dati
- Progetto dei sottosistemi dati
- Implementazione come FSM dei sottosistemi controllo.

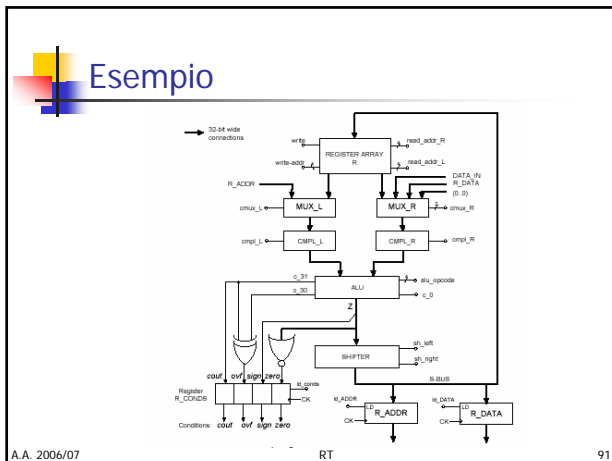
A.A. 2006/07 03\_II progetto di sistemi a livello RT 89



### Sottosistemi dati: componenti

- Moduli di memorizzazione
- Moduli funzionali (operatori)
- Inrerconnessioni (datapath): fili, switch
- Punti di controllo: connessione in ingresso dei segnali di controllo ai moduli
- Punti di condizione: connessione in uscita dei segnali di condizione dai moduli.

A.A. 2006/07 03\_II progetto di sistemi a livello RT 90



### Moduli di memorizzazione

- Registri singoli, con connessioni e controlli separati, maggiore concorrenza, più complessa l'interconnessione
- array di registri che condividono connessioni e controlli:
  - register file
  - RAM
- combinazioni di registri singoli e array di registri.

03\_II progetto di sistemi a livello RT 92

### Register File

Array di registri:

- operazioni di lettura/scrittura multiple e contemporanee

Esempio:

- 2 letture contemporanee e 1 scrittura
- controllo di scrittura ma non di lettura

In generale i register file sono piccoli (da 8 a 32) e veloci (comparabili con la ALU).

03\_II progetto di sistemi a livello RT 93

### Esempio

```

USE WORK.BitDefs_pkg.ALL;
ENTITY reg_file IS
    GENERIC (n: NATURAL:=16;
             p: NATURAL:= 8;
             k: NATURAL:= 3;
             Td: TIME:= 5 ns);
    PORT (X : IN UNSIGNED(n-1 DOWNTO 0);
          WA : IN UNSIGNED(k-1 DOWNTO 0);
          RA1 : IN UNSIGNED(k-1 DOWNTO 0);
          RA2 : IN UNSIGNED(k-1 DOWNTO 0);
          Z1,Zr: OUT UNSIGNED(n-1 DOWNTO 0);
          Wr : IN BIT; --
          clk : IN BIT);
END reg_file;
    
```

03\_II progetto di sistemi a livello RT 94

```

ARCHITECTURE behavioral OF reg_file IS
    SUBTYPE WordT IS UNSIGNED(n-1 DOWNTO 0);
    TYPE StorageT IS ARRAY(0 TO p-1) OF WordT;
    SIGNAL RF: StorageT;
    BEGIN
    PROCESS (clk)
    BEGIN
        IF (clk'EVENT AND clk = '1') AND (Wr = '1') THEN
            RF(CONV_INTEGER(WA)) <= X;
        END IF;
    END PROCESS;

    PROCESS (RA1,RAr,RF)
    BEGIN
        Z1 <= RF(CONV_INTEGER(RA1)) AFTER Td;
        Zr <= RF(CONV_INTEGER(RAr)) AFTER Td;
    END PROCESS;
    END behavioral;
    
```

03\_II progetto di sistemi a livello RT 95

### RAM

- solo 1 operazione di lettura/scrittura alla volta
- controllo di scrittura e di lettura
- maggior ritardo, segnale di scrittura come clock.

03\_II progetto di sistemi a livello RT 96

### Esempio

```

ENTITY ram IS
  GENERIC(n: NATURAL:= 16;
    p: NATURAL:=256;
    k: NATURAL:= 8;
    Td: TIME:= 40 ns);
  PORT(X : IN UNSIGNED(n-1 DOWNT0);
    A : IN UNSIGNED(k-1 DOWNT0);
    Z : OUT UNSIGNED(n-1 DOWNT0);
    Rd,Wr: IN BIT;
    Clk : IN BIT);
END ram;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 97

```

ARCHITECTURE behavioral OF ram IS
  SUBTYPE WordT IS UNSIGNED(n-1 DOWNT0);
  TYPE StorageT IS ARRAY(0 TO p-1) OF WordT;
  SIGNAL Memory: StorageT;
BEGIN
  PROCESS (Clk)
  BEGIN
    IF (Clk'EVENT AND Clk = '1') AND (Wr = '1') THEN
      Memory(CONV_INTEGER(A)) <= X;
    END IF;
  END PROCESS;
  PROCESS (Rd,Memory)
  BEGIN
    IF (Rd = '1') THEN
      Z <= Memory(CONV_INTEGER(A)) AFTER Td;
    END IF;
  END PROCESS;
END behavioral;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 98

### Moduli funzionali

```

CASE op_sel IS
  WHEN F1 => z_out <= x_in op1 y_in AFTER delay;
  WHEN F2 => z_out <= x_in op2 y_in AFTER delay;
  ....
END CASE;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 99

### Interconnessioni (datapath)

Tipo di connessione:

- diretta (fili)
- indiretta (attraverso switch)

Parallelismo:

- 1 bit alla volta (seriale)
- più bit alla volta (parallelo)

Direzionalità:

- monodirezionale (sorgente e destinazione fisse e non scambiabili)
- bidirezionale

A.A. 2006/07 03\_II progetto di sistemi a livello RT 100

Condivisione:

- dedicato: sorgente e destinazione uniche
- condiviso o bus: sorgente e destinazione multiple, una sola trasmissione alla volta

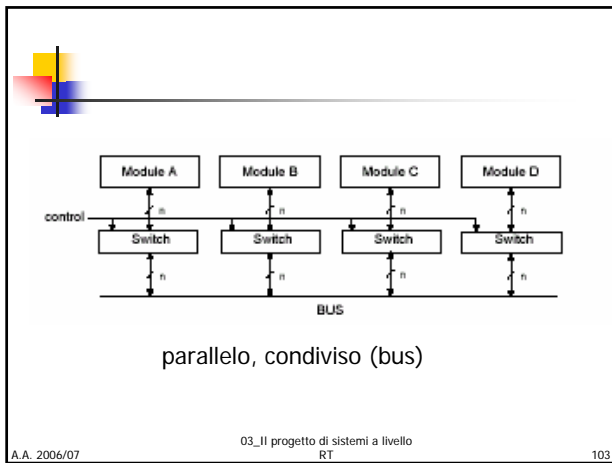
Diretto/indiretto:

- diretto se esiste connessione diretta
- indiretto se esiste connessione attraverso componente

A.A. 2006/07 03\_II progetto di sistemi a livello RT 101

### Esempi

A.A. 2006/07 03\_II progetto di sistemi a livello RT 102



### Switch

Abilitazioni dei cammini in datapath condivisi. Implementati con:

- porte tri-state
- multiplexer.

porte tri-state

A.A. 2006/07 03\_II progetto di sistemi a livello RT 104

### multiplexer

```

PROCESS (Xp1,...,X0,S)
BEGIN
CASE S IS
WHEN "0..0" => Z <= X0;
WHEN "0..1" => Z <= X1;
....
WHEN "1..1" => Z <= Xp;
END CASE;
END PROCESS;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 105

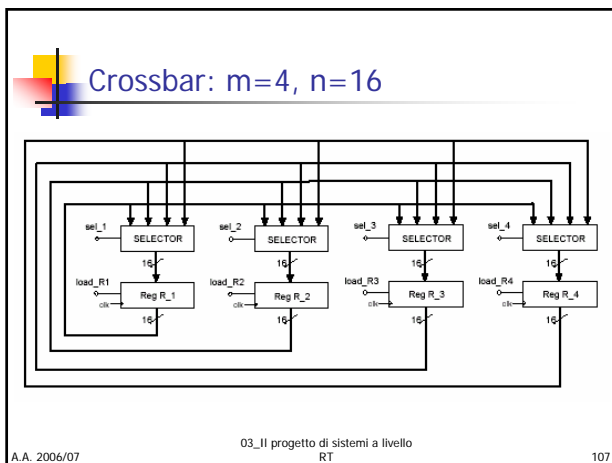
### Tipi di datapath

m registri di n bit, datapath per poter trasferire tra qualsiasi coppia di registri.

Casi estremi:

- interconnessione completa (crossbar): m trasferimenti simultanei con m selettori e m log m segnali di selezione
- bus: solo una sorgente alla volta è connessa al bus

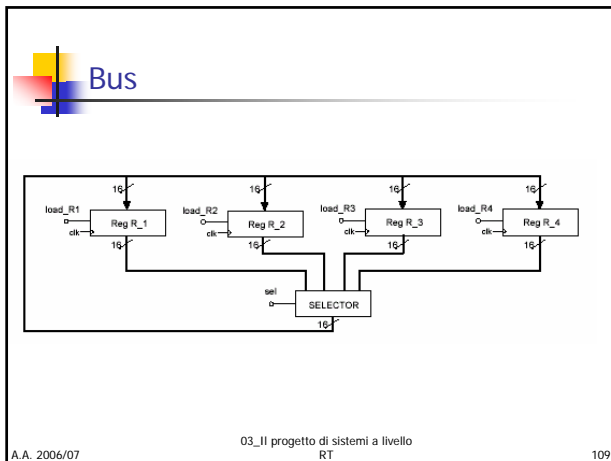
A.A. 2006/07 03\_II progetto di sistemi a livello RT 106



```

SIGNAL R_1,R_2,R_3,R_4: BIT_VECTOR(15 DOWNT0);
PROCESS (clk)
VARIABLE Sel1,Sel2,Sel3,Sel4: BIT_VECTOR(15 DOWNT0);
BEGIN
CASE sel_1 IS
WHEN "00" => Sel1:= R1;
WHEN "01" => Sel1:= R2;
WHEN "10" => Sel1:= R3;
WHEN "11" => Sel1:= R4;
END CASE;
....
IF (clk='1') THEN
IF (load_R1 = '1') THEN R_1 <= Sel1; ENDIF;
IF (load_R2 = '1') THEN R_2 <= Sel1; ENDIF;
IF (load_R3 = '1') THEN R_3 <= Sel1; ENDIF;
IF (load_R4 = '1') THEN R_4 <= Sel4; ENDIF;
END IF;
END PROCESS;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 108



```

SIGNAL R_1,R_2,R_3,R_4: BIT_VECTOR(15 DOWNTO 0);
PROCESS (clk)
VARIABLE Sel_out: BIT_VECTOR(15 DOWNTO 0);
BEGIN
CASE sel IS
  WHEN "00" => Sel_out:= R_1;
  WHEN "01" => Sel_out:= R_2;
  WHEN "10" => Sel_out:= R_3;
  WHEN "11" => Sel_out:= R_4;
END CASE;
IF (clk='1') THEN
  IF (load_R1 = '1') THEN R_1 <= Sel_out; ENDIF;
  IF (load_R2 = '1') THEN R_2 <= Sel_out; ENDIF;
  IF (load_R3 = '1') THEN R_3 <= Sel_out; ENDIF;
  IF (load_R4 = '1') THEN R_4 <= Sel_out; ENDIF;
END IF;
END PROCESS;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 110

- ### Sottosistema controllo
- FSM:
- input: segnali di controllo dall'esterno e segnali di condizione dal sottosistema dati
  - output: segnali di controllo
  - uno stato per ogni statement nella sequenza di register-transfer
  - la funzione di transizione corrisponde alla sequenzializzazione
  - l'uscita per ogni stato corrisponde ai segnali che controllano l'attività in quello stato.
- A.A. 2006/07 03\_II progetto di sistemi a livello RT 111

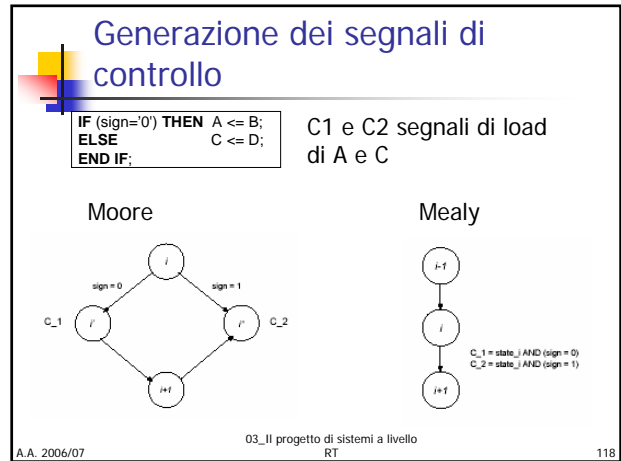
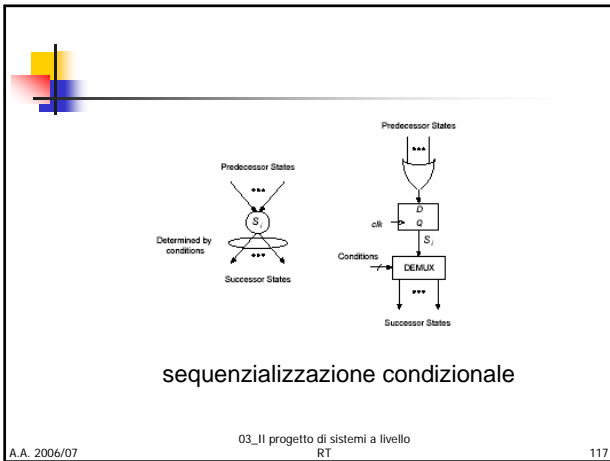
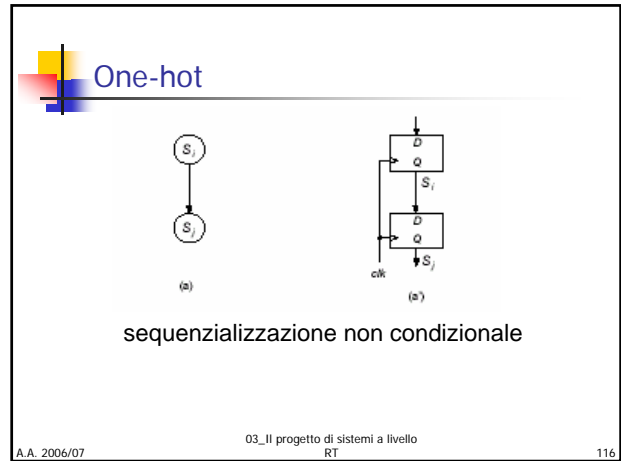
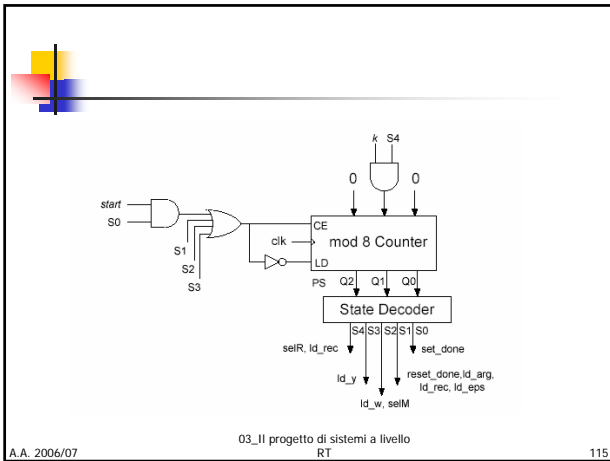
- ### Assegnazione degli stati
- casuale
  - basata su contatore
  - one-hot.
- A.A. 2006/07 03\_II progetto di sistemi a livello RT 112

- ### Uso di contatori
- I grafi di esecuzione group-sequential hanno:
- sequenzializzazione non condizionale (stato con unico successore)
  - sequenzializzazione condizionale (più successori possibili in funzione di una condizione)
- Prevale la sequenzializzazione non condizionale, implementabile con un contatore
- A.A. 2006/07 03\_II progetto di sistemi a livello RT 113

### Esempio

PS	Condition	NS	Count Enable	Parallel Load	Parallel Inputs	Active Control Signals
S0	start = 0	S0	0	1	000	set_done
S0	start = 1	S1	1	0	-	set_done
S1	-	S2	1	0	-	reset_done, ld_arg, ld_rec, ld_eps
S2	-	S3	1	0	-	ld_w, selM
S3	-	S4	1	0	-	ld_y
S4	k = 1	S2	0	1	010	ld_rec, selR
S4	k = 0	S0	0	1	000	ld_rec, selR

A.A. 2006/07 03\_II progetto di sistemi a livello RT 114



### Esempio incdec

Specifica: calcolare

INPUTS:  $x, y \in \{-128, \dots, 127\}$

OUTPUT:  $z \in \{-256, \dots, 508\}$

FUNCTION:  $z = \begin{cases} 4\lceil(x + |y|)/2\rceil & \text{if } x < |y| \\ 4x & \text{otherwise} \end{cases}$

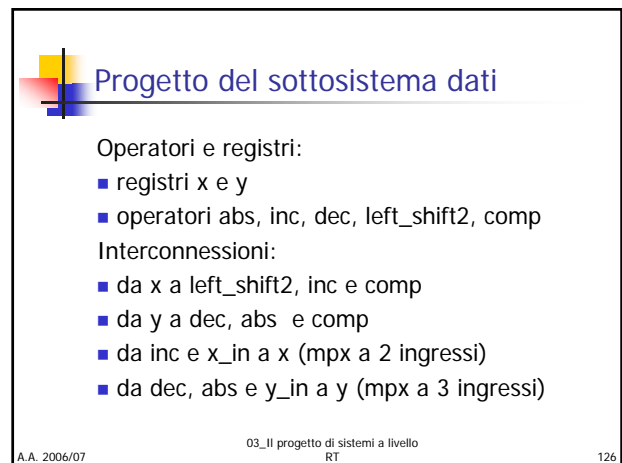
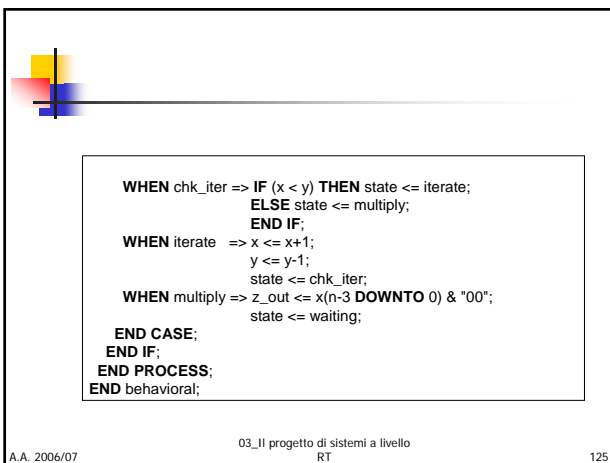
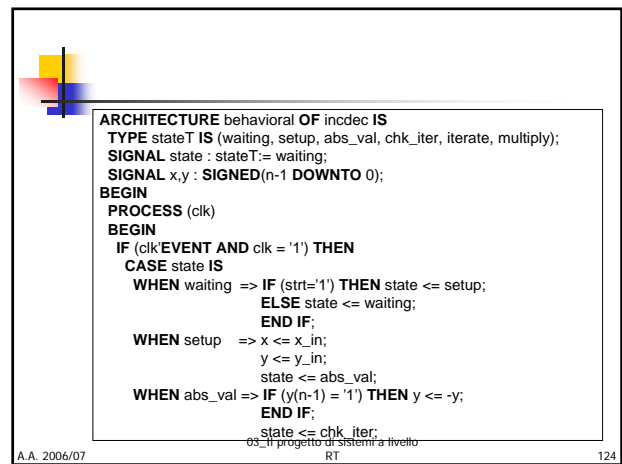
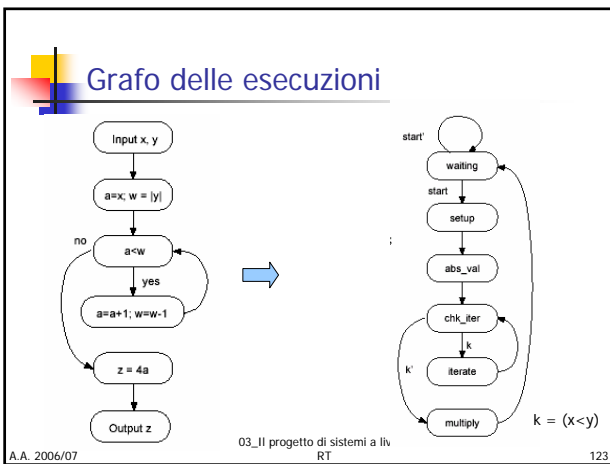
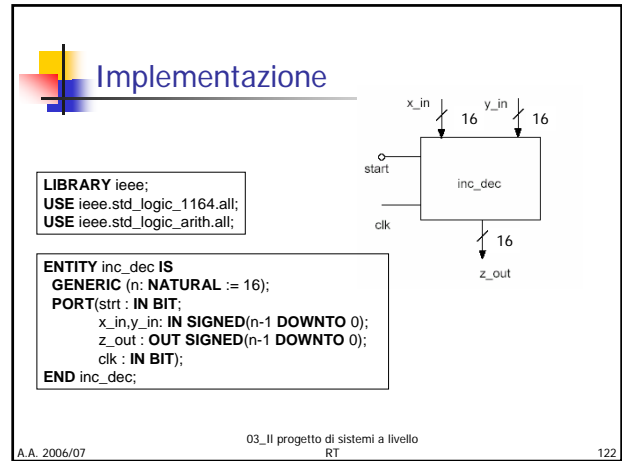
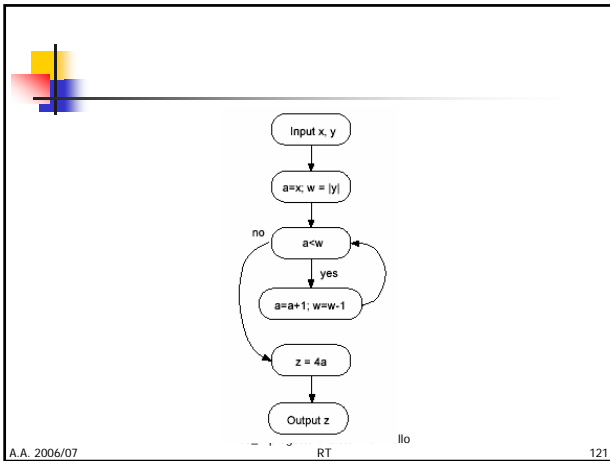
senza usare sommatore a due operandi.

A.A. 2006/07 03\_II progetto di sistemi a livello RT 119

$$a = (x + |y|)/2 = x + (|y| - x)/2$$

- inizializzare a a x
- incrementare a e decrementare |y| mentre  $a < |y|$

A.A. 2006/07 03\_II progetto di sistemi a livello RT 120



Punti di controllo:

Operation	Control Points
load register x	ldX
load register y	ldY
select input to x (1 bit)	selX
select input to y (2 bit)	selY

Condizioni:

$$k = \begin{cases} 1 & \text{if } x < y \\ 0 & \text{otherwise} \end{cases}$$

A.A. 2006/07 03\_II progetto di sistemi a livello RT 127

### Schema a blocchi

A.A. 2006/07 03\_II progetto di sistemi a livello RT 128

### Descrizione structural

```

ARCHITECTURE structural OF incdec_data IS
SIGNAL inc_out,dec_out,abs_out,lft_out,mux1_out: BIT_VECTOR(n-1 DOWNTO 0);
SIGNAL mux2_out,xreg_out,yreg_out,zero_32: BIT_VECTOR(n-1 DOWNTO 0);
BEGIN
x: ENTITY reg PORT MAP (mux1_out, ldX, xreg_out, clk);
y: ENTITY reg PORT MAP (mux2_out, ldY, yreg_out, clk);
inc: ENTITY incrementer PORT MAP (xreg_out, inc_out);
dec: ENTITY decrementer PORT MAP (yreg_out, dec_out);
ab: ENTITY absolute PORT MAP (yreg_out, abs_out);
mx1: ENTITY mux2 PORT MAP (x_in, inc_out, selX, mux1_out);
mx2: ENTITY mux4 PORT MAP (y_in, dec_out, abs_out, zero_32,
selY, mul2_out);
lft2: ENTITY lft_shift2 PORT MAP (xreg_out, lft_out);
comp: ENTITY less_than PORT MAP (xreg_out, yreg_out, k);
END structural;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 129

### Descrizione behavioral

```

USE WORK.BitDefs_pkg.ALL;
ENTITY data_subsystem IS
GENERIC (n: NATURAL:= 16);
PORT (x_in, y_in: IN BIT_VECTOR(n-1 DOWNTO 0);
ldX,ldY: IN BIT;
selX: IN BIT;
selY: IN BIT_VECTOR(1 DOWNTO 0);
k: OUT BIT;
z_out: OUT BIT_VECTOR(n-1 DOWNTO 0);
clk: IN BIT);
END data_subsystem ;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 130

### Descrizione behavioral

```

ARCHITECTURE behavioral OF data_subsystem IS
SIGNAL x,y,z: BIT_VECTOR(n-1 DOWNTO 0);
BEGIN
PROCESS (clk)
BEGIN
IF (clk = '1') THEN
IF (ldX = '1') AND (selX = '0') THEN x <= x_in; END IF;
IF (ldX = '1') AND (selX = '1') THEN x <= inc(x); END IF;
IF (ldY = '1') AND (selY = "00") THEN y <= y_in; END IF;
IF (ldY = '1') AND (selY = "01") THEN y <= dec(y); END IF;
IF (ldY = '1') AND (selY = "10") THEN y <= ABS(y); END IF;
IF (x < y) THEN k <= '1';
ELSE k <= '0';
END IF;
z_out <= shift_left(x,2);
END IF;
END PROCESS;
END behavioral;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 131

### Descrizione strutturale interfaccia

```

ARCHITECTURE structural OF incdec IS
SIGNAL selY: BIT_VECTOR(1 DOWNTO 0);
SIGNAL ldX,ldY,selX: BIT;
SIGNAL k: BIT;
BEGIN
datasub: ENTITY incdec_data PORT MAP (x_in,y_in,ldX,ldY,selX,selY,k,z_out,clk);
ctrlsub: ENTITY incdec_ctrl PORT MAP (start,k,ldX,ldY,selX,selY,clk);
END structural;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 132

### Implementazione dei moduli

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY absolute IS
  PORT (x_in : IN SIGNED;
         z_out : OUT SIGNED);
END absolute;

ARCHITECTURE behavioral OF absolute IS
BEGIN
  PROCESS (x_in)
  BEGIN
    IF x_in(x_in'LEFT) = '1' THEN z_out <= -x_in;
    ELSE z_out <= x_in;
    END IF;
  END PROCESS;
END behavioral;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 133

### Progetto del sottosistema controllo

State	ldX	ldY	selX	selY	Next state
waiting	0	0	-	-	setup, waiting
setup	1	1	0	00	abs_val
abs_val	0	1	-	01	chk_iter
chk_iter	0	0	-	-	iterate, multiply
iterate	1	1	1	10	chk_iter
multiply	0	0	-	-	waiting

Tabella delle funzioni di controllo

A.A. 2006/07 03\_II progetto di sistemi a livello RT 134

### Descrizione behavioral

```

ENTITY incdec_ctrl IS
  PORT (strt,k, clk : IN BIT;
         ldX, ldY : OUT BIT;
         selX : OUT BIT;
         selY : OUT BIT_VECTOR(1 DOWNTO 0));
END incdec_ctrl ;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 135

```

ARCHITECTURE behavioral OF incdec_ctrl IS
  TYPE stateT IS (waiting,setup,abs_val,chk_iter,iterate,multiply);
  SIGNAL state : stateT:= waiting;
BEGIN
  PROCESS (clk)
  BEGIN
    IF (clk'EVENT AND clk = '1') THEN
      CASE state IS
        WHEN waiting => IF (strt = '1') THEN state <= setup; END IF;
        WHEN setup => state <= abs_val ;
        WHEN abs_val => state <= chk_iter;
        WHEN chk_iter => IF (k = '1') THEN state <= iterate ;
                       ELSE state <= multiply;
                       END IF;
        WHEN iterate => state <= chk_iter;
        WHEN multiply => state <= waiting ;
      END CASE;
    END IF;
  END PROCESS;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 136

```

PROCESS (state)
  VARIABLE ctrls : BIT_VECTOR(4 DOWNTO 0);
BEGIN
  CASE state IS
    WHEN waiting => ctrls(4 downto 3):= "00" ;
    WHEN setup => ctrls := "11000";
    WHEN abs_val => ctrls(4 downto 3):= "01" ;
                    ctrls(1 downto 0):= "01" ;
    WHEN chk_iter => ctrls(4 downto 3):= "00" ;
    WHEN iterate => ctrls := "11110";
    WHEN multiply => ctrls(4 downto 3):= "00" ;
  END CASE;
  ldX <= ctrls(4); ldY <= ctrls(3);
  selX <= ctrls(2); selY <= ctrls(1 DOWNTO 0);
END PROCESS;
END behavioral;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 137

### Implementazione sottosistema controllo con codifica one-hot

A.A. 2006/07 03\_II RT 138

## Specifica e implementazione di un microcomputer

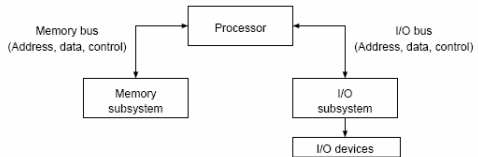
cap. 15 Introduction to Digital Systems - Wiley



Milos Ercegovac, Tomas Lang, Jaime Moreno  
*University of California*

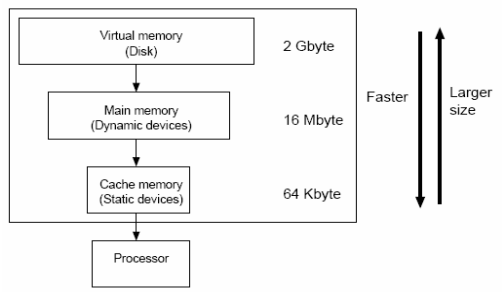
## Architettura di un microcomputer

- processore
- sottosistema memoria
- sottosistema di I/O



03\_II progetto di sistemi a livello RT

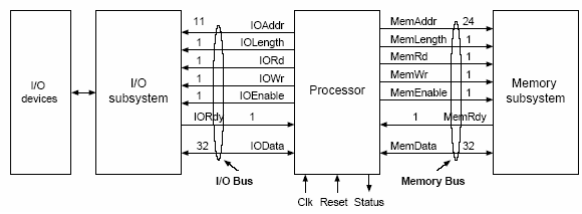
## Gerarchia di memoria



03\_II progetto di sistemi a livello RT

## Specifica del sistema

### Struttura



03\_II progetto di sistemi a livello RT

- address bus:
  - MemAddr: 24 bit,  $2^{24}$  locazioni di memoria
  - IOAddr: 11 bit,  $2^{11}$  porte di I/O
- segnali di controllo (1 bit):
  - lettura/scrittura: MemRd, MemWr, MemEnable, IORd, IOWr, IOEnable
  - lunghezza operando: MemLength, IOLength

03\_II progetto di sistemi a livello RT

- segnali di stato (1 bit):
  - memoria: MemRdy
  - I/O: IORdy
  - Status: attività all'interno del ciclo istruzione
- data bus (32 bit):
  - MemData
  - IOData

03\_II progetto di sistemi a livello RT

### Descrizione strutturale

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

PACKAGE comp_pkg IS
  SUBTYPE WordT IS STD_LOGIC_VECTOR(31 DOWNTO 0);
  SUBTYPE MAddrT IS STD_LOGIC_VECTOR(23 DOWNTO 0);
  SUBTYPE IOAddrT IS STD_LOGIC_VECTOR(10 DOWNTO 0);
  SUBTYPE ByteT IS STD_LOGIC_VECTOR( 7 DOWNTO 0);
  TYPE StatusT IS (undef, p_reset, fetch, execute, memop, ioop);

  FUNCTION get_carry(RA_Data, RB_Data, Imm, Opcode: STD_LOGIC_VECTOR)
    RETURN STD_LOGIC;
  FUNCTION get_ovf (RA_Data, RB_Data, Imm, Opcode: STD_LOGIC_VECTOR)
    RETURN STD_LOGIC;
  FUNCTION get_cc (RA_Data, RB_Data, Opcode: STD_LOGIC_VECTOR)
    RETURN STD_LOGIC_VECTOR;
END comp_pkg;
    
```

A.A. 2006/07 03\_Il progetto di sistemi a livello RT 145

```

PACKAGE BODY comp_pkg IS
  FUNCTION get_carry(RA_Data, RB_Data, Imm, Opcode: STD_LOGIC_VECTOR)
    RETURN STD_LOGIC
  IS VARIABLE cy: STD_LOGIC:= '0';
  BEGIN
    -- description of carry generation included here
    RETURN(cy);
  END get_carry;
  FUNCTION get_ovf (RA_Data, RB_Data, Imm, Opcode: STD_LOGIC_VECTOR)
    RETURN STD_LOGIC
  IS VARIABLE ovf: STD_LOGIC:= '0';
  BEGIN
    -- description of overflow generation included here
    RETURN(ovf);
  END get_ovf;
  FUNCTION get_cc (RA_Data, RB_Data, Opcode: STD_LOGIC_VECTOR)
    RETURN STD_LOGIC_VECTOR
  IS VARIABLE cc: STD_LOGIC_VECTOR(3 DOWNTO 0):= "0000";
  BEGIN
    -- description of cc generation included here
    RETURN(cc);
  END get_cc;
END comp_pkg;
    
```

A.A. 2006/07 03\_Il progetto di sistemi a livello RT 146

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE WORK.ALL, WORK.comp_pkg.ALL;
ENTITY Computer IS
  PORT (Reset, Clk : IN STD_LOGIC);
END Computer;
ARCHITECTURE structural OF Computer IS
  SIGNAL MemAddr : MAddrT ;
  SIGNAL MemLength, MemRd : STD_LOGIC;
  SIGNAL MemWr, MemEnable : STD_LOGIC;
  SIGNAL MemRdy : STD_LOGIC;
  SIGNAL MemData : WordT ;
  SIGNAL IOAddr : IOAddrT ;
  SIGNAL IOLength, IORd : STD_LOGIC;
  SIGNAL IOWr, IOEnable : STD_LOGIC;
  SIGNAL IORdy : STD_LOGIC;
  SIGNAL IOData : WordT ;
  SIGNAL Status : StatusT;
    
```

A.A. 2006/07 03\_Il progetto di sistemi a livello RT 147

```

BEGIN
  U1: ENTITY Memory
    PORT MAP (MemAddr, MemLength, MemRd, MemWr, MemEnable,
              MemRdy, MemData);

  U2: ENTITY IO
    PORT MAP (IOAddr, IOLength, IORd, IOWr, IOEnable, IORdy, IOData);

  U3: ENTITY Processor
    PORT MAP (MemAddr, MemData, MemLength, MemRd, MemWr,
              MemEnable, MemRdy, IOAddr, IOData, IOLength, IORd,
              IOWr, IOEnable, IORdy, Status, Reset, Clk);
END structural;
    
```

A.A. 2006/07 03\_Il progetto di sistemi a livello RT 148

### Sottosistema memoria

Accesso alla memoria:

- singolo byte
- parola da 32 bit

A.A. 2006/07 03\_Il progetto di sistemi a livello RT 149

A.A. 2006/07 03\_Il progetto di sistemi a livello RT 150

### Entity declaration

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE WORK.comp_pkg.ALL;

ENTITY Memory IS
  PORT (Addr : IN MAddrT ; -- memory address bus
    Length : IN STD_LOGIC; -- byte/word operand
    Rd, Wr : IN STD_LOGIC; -- access control signals
    Enable : IN STD_LOGIC; -- enable signal
    Rdy : OUT STD_LOGIC; -- access completion signal
    Data : INOUT WordT ); -- memory data bus
END Memory;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 151

### Descrizione behavioral

```

LIBRARY ieee;
USE ieee.std_logic_unsigned.ALL;
ARCHITECTURE behavioral OF Memory IS
  CONSTANT Tmem : TIME := 8 ns;
  CONSTANT Td : TIME := 200 ps;
  CONSTANT Tsu : TIME := 200 ps;
BEGIN
  PROCESS (Rd, Wr, Enable)
    CONSTANT byte_l : STD_LOGIC := '0';
    CONSTANT word_l : STD_LOGIC := '1';
    CONSTANT MaxMem : NATURAL := 16#FFFFFF#;
    TYPE MemArrayT IS ARRAY(0 TO MaxMem-1) OF ByteT;
    VARIABLE Mem : MemArrayT;
    VARIABLE tAddr : NATURAL;
    VARIABLE tData : WordT ;
    VARIABLE tCtrl : STD_LOGIC_VECTOR(2 DOWNTO 0);
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 152

```

BEGIN
  tCtrl := Rd & Wr & Enable; -- group signals for simpler decoding
  CASE tCtrl IS
    -- output to tri-state
    WHEN "000" => Data <= (OTHERS => 'Z') AFTER Td;
    -- write access;

    WHEN "011" =>
      -- indicate module busy
      Rdy <= '0' AFTER Td, '1' AFTER Tmem;
    IF (Length = byte_l) THEN -- read address
      tAddr := CONV_INTEGER(Addr); -- bit-vector to integer
      -- from pkg std_logic_unsigned

    ELSE
      tAddr := CONV_INTEGER(Addr(23 DOWNTO 2) & "00");
    END IF;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 153

```

  CASE Length IS
    WHEN byte_l => Mem(tAddr) := (Data( 7 DOWNTO 0));
    WHEN word_l => Mem(tAddr) := (Data( 7 DOWNTO 0));
      Mem(tAddr+1) := (Data(15 DOWNTO 8));
      Mem(tAddr+2) := (Data(23 DOWNTO 16));
      Mem(tAddr+3) := (Data(31 DOWNTO 24));
    WHEN OTHERS => NULL;
  END CASE;
  WHEN "101" =>
    -- read access
    -- indicate module busy
    Rdy <= '0' AFTER Td, '1' AFTER Tmem;
  IF (Length = byte_l) THEN -- read address
    tAddr := CONV_INTEGER(Addr); -- bit-vector to integer
  ELSE
    tAddr := CONV_INTEGER(Addr(23 DOWNTO 2) & "00");
  END IF;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 154

```

  CASE Length IS
    WHEN byte_l => tData( 7 DOWNTO 0) := (Mem(tAddr));
    WHEN word_l => tData( 7 DOWNTO 0) := (Mem(tAddr));
      tData(15 DOWNTO 8) := (Mem(tAddr+1));
      tData(23 DOWNTO 16) := (Mem(tAddr+2));
      tData(31 DOWNTO 24) := (Mem(tAddr+3));
    WHEN OTHERS => NULL;
  END CASE;
  Data <= tData AFTER Tmem; -- deliver data
  WHEN OTHERS => NULL; -- memory not enabled
END CASE;
END PROCESS;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 155

```

  -- timing verifications
  ASSERT NOT (Rd'EVENT AND Rd='1' AND NOT Addr'STABLE(Tsu))
  REPORT "Read address setup time violation";
  ASSERT NOT (Rd'EVENT AND Rd='1' AND NOT Enable'STABLE(Tsu))
  REPORT "Read enable setup time violation";
  ASSERT NOT (Wr'EVENT AND Wr='1' AND NOT Addr'STABLE(Tsu))
  REPORT "Write address setup time violation";
  ASSERT NOT (Wr'EVENT AND Wr='1' AND NOT Enable'STABLE(Tsu))
  REPORT "Write enable setup time violation";
END behavioral;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 156

### Sottosistema I/O

Interfacce ai dispositivi per trasferimento I/O di dati:

- 2048 porte

A.A. 2006/07 03\_II progetto di sistemi a livello RT 157

### Entity declaration

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE WORK.comp_pkg.ALL;
ENTITY IO IS
  PORT (Addr : IN IOAddrT ; -- I/O address bus
        Length : IN STD_LOGIC; -- byte/word control
        Rd, Wr : IN STD_LOGIC; -- I/O access control
        Enable : IN STD_LOGIC; -- I/O enable control
        Rdy : OUT STD_LOGIC; -- I/O completion signal
        Data : INOUT WordT ); -- I/O data bus
END IO;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 158

### Descrizione behavioral

Simile a quella del sottosistema memoria, con spazio di indirizzamento diverso.

A.A. 2006/07 03\_II progetto di sistemi a livello RT 159

### Processore

Stato del processore:

- 32 registri general-purpose da 32 bit (R0...R31)
- Program counter PC da 24 bit
- Condition register CR da 4 bit
- Instruction Register IR da 32 bit

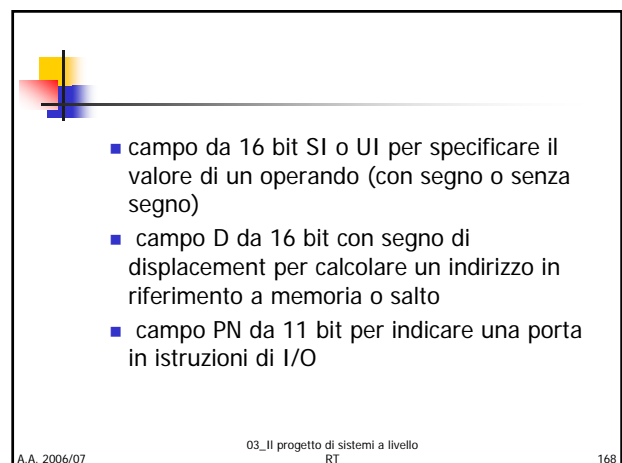
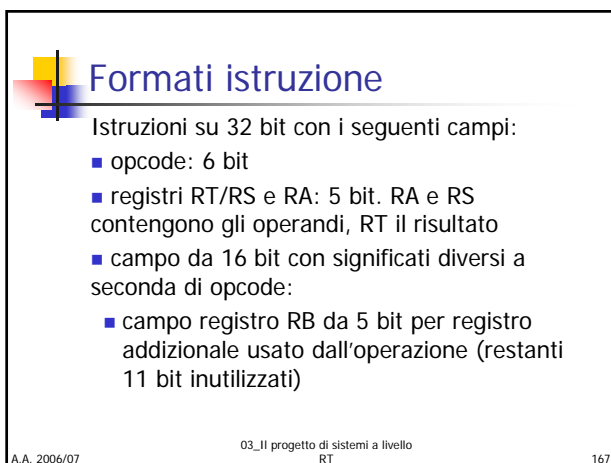
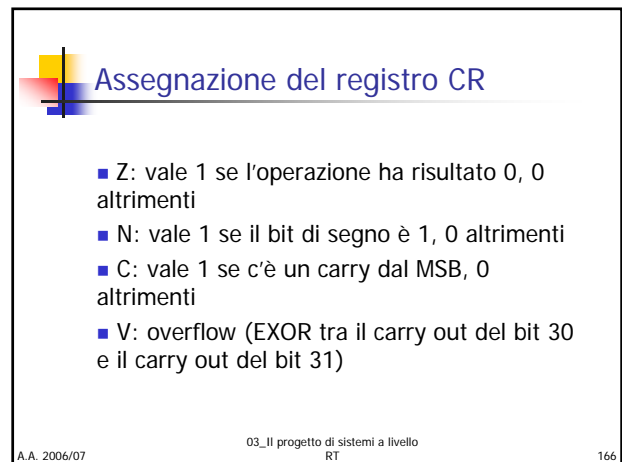
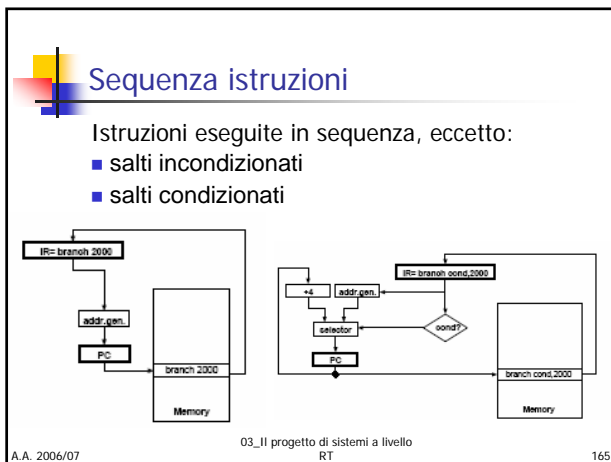
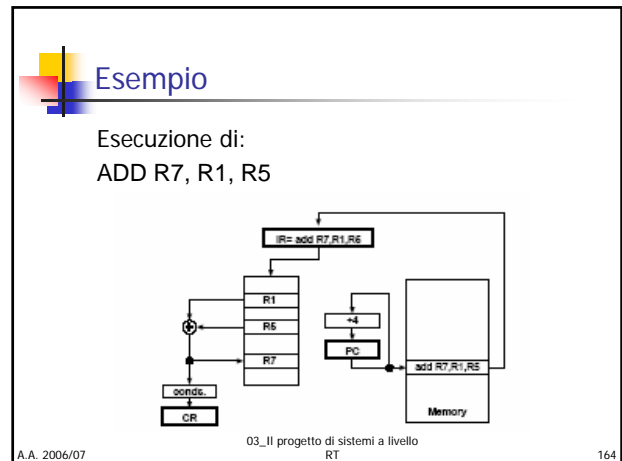
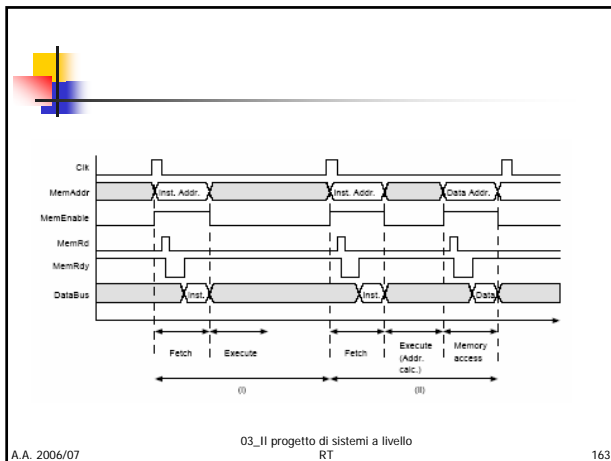
A.A. 2006/07 03\_II progetto di sistemi a livello RT 160

A.A. 2006/07 03\_II progetto di sistemi a livello RT 161

### Ciclo istruzione di base:

- se RESET è a 1, azzerare tutti i registri
- se CLK è a 1, eseguire il ciclo fetch-execute

A.A. 2006/07 03\_II progetto di sistemi a livello RT 162



## Tipi di istruzioni

Istruzioni:

- unarie (1 solo operando)
- binarie (2 operandi)
- con celle di memoria
- I/O
- salto

A.A. 2006/07 03\_II progetto di sistemi a livello RT 169

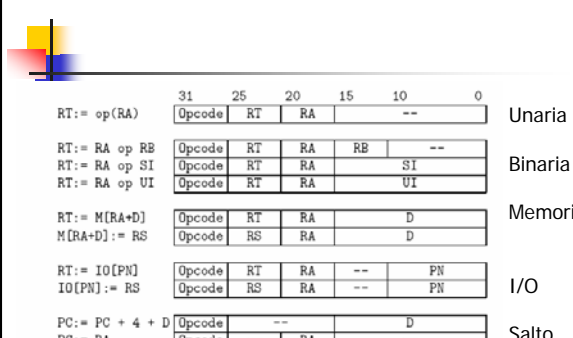


Figure 15.9: INSTRUCTION FORMATS.

A.A. 2006/07 03\_II progetto di sistemi a livello RT 170

## Instruction set

Name	Opcode	Function	CR	Assembly Language
No-op	000000	no operation		nop
NOT	000010	RT := not(RA)	Y	not RT,RA
Left shift	000100	RT := lshift(RA)	Y	lah RT,RA
Right shift	000110	RT := rshift(RA)	Y	rsh RT,RA
Left rotate	001000	RT := lrot(RA)	Y	lrt RT,RA
Right rot.	001010	RT := rrot(RA)	Y	rrt RT,RA
Add	010000	RT := RA + RB	Y	add RT,RA,RB
Add immed.	010001	RT := RA + SI	Y	adi RT,RA,SI
Subtract	010010	RT := RA - RB	Y	sub RT,RA,RB
Sub. immed.	010011	RT := RA - SI	Y	sbi RT,RA,SI
AND	010100	RT := RA and RB	Y	and RT,RA,RB
AND immed.	010101	RT := RA and UI	Y	ani RT,RA,UI
OR	010110	RT := RA or RB	Y	or RT,RA,RB
OR immed.	010111	RT := RA or UI	Y	ori RT,RA,UI
XOR	011000	RT := RA xor RB	Y	xor RT,RA,RB
XOR immed.	011001	RT := RA xor UI	Y	xri RT,RA,UI

A.A. 2006/07 03\_II progetto di sistemi a livello RT 171

Name	Opcode	Function	CR	Assembly Language
Load byte	100000	RT(7 to 0) := Mem(RA+D,1)		ldb RT,D(RA)
Load word	100001	RT(31 to 0) := Mem(RA+D,4)		ldw RT,D(RA)
Store byte	100010	Mem(RA+D,1) := RS(7 to 0)		stb RS,D(RA)
Store word	100011	Mem(RA+D,4) := RS(31 to 0)		stw RS,D(RA)
I/O Rd byte	100100	RT(7 to 0) := IO(PN,1)		irb RT,PN
I/O Rd word	100101	RT(31 to 0) := IO(PN,4)		irw RT,PN
I/O Wr byte	100110	IO(PN,1) := RS(7 to 0)		iwb RS,PN
I/O Wr word	100111	IO(PN,4) := RS(31 to 0)		iww RS,PN

A.A. 2006/07 03\_II progetto di sistemi a livello RT 172

Name	Opcode	Function	CR	Assembly Language
Branch	111000	PC := PC + 4 + D		br D
Branch indirect	111001	PC := RA		bri RA
Branch if N=0	110000	If N=0 then PC := PC+4+D		brp D
Branch if N=1	110001	If N=1 then PC := PC+4+D		brn D
Branch if Z=0	110010	If Z=0 then PC := PC+4+D		bnz D
Branch if Z=1	110011	If Z=1 then PC := PC+4+D		brz D
Branch if C=0	110100	If C=0 then PC := PC+4+D		bnc D
Branch if C=1	110101	If C=1 then PC := PC+4+D		brc D
Branch if V=0	110110	If V=0 then PC := PC+4+D		bnv D
Branch if V=1	110111	If V=1 then PC := PC+4+D		brv D

A.A. 2006/07 03\_II progetto di sistemi a livello RT 173

## Rappresentazione dei dati

Un dato su 32 bit può rappresentare:

- un booleano
- un intero con o senza segno in complemento a 2
- un intero senza segno per indirizzi
- Le operazioni di I/O o di riferimento alla memoria possono usare dati su 8 bit interi senza segno o booleani

A.A. 2006/07 03\_II progetto di sistemi a livello RT 174

### Entity declaration

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE WORK.comp_pkg.ALL;

ENTITY processor IS
  PORT (MemAddr : OUT MAddrT ; -- memory address bus
        MemData : INOUT WordT ; -- data bus to/from memory
        MemLength : OUT STD_LOGIC; -- memory operand length
        MemRd : OUT STD_LOGIC; -- memory read control signal
        MemWr : OUT STD_LOGIC; -- memory write control signal
        MemEnable : OUT STD_LOGIC; -- memory enable signal
        MemRdy : IN STD_LOGIC; -- memory completion signal
  );
  
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 175

```

        IOAddr : OUT IOAddrT ; -- I/O address bus
        IOData : INOUT WordT ; -- data bus to/from I/O
        IOLength : OUT STD_LOGIC; -- I/O operand length
        IORd : OUT STD_LOGIC; -- I/O read control signal
        IOWr : OUT STD_LOGIC; -- I/O write control signal
        IOEnable : OUT STD_LOGIC; -- memory enable signal
        IORdy : IN STD_LOGIC; -- I/O completion signal
        Status : OUT StatusT ; -- processor status signal
        Reset : IN STD_LOGIC; -- reset signal
        Clk : IN STD_LOGIC; -- clock signal
END processor;
  
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 176

### Descrizione behavioral

```

LIBRARY ieee;
USE ieee.std_logic_arith.all; -- use definitions and operations
USE ieee.std_logic_signed.all; -- on signed values

ARCHITECTURE behavioral OF processor IS
  -- registers (processor state)
  TYPE RegFileT IS ARRAY(0 to 31) OF WordT;
  SIGNAL GPR : RegFileT ; -- general registers
  SIGNAL PC : MAddrT ; -- Program Counter register
  SIGNAL CR : STD_LOGIC_VECTOR(3 DOWNTO 0); -- Condition Register
  SIGNAL IR : STD_LOGIC_VECTOR(31 DOWNTO 0); -- Instruction register

  -- signals used by output function
  SIGNAL Phase : StatusT ; -- instr. cycle phase
  SIGNAL tMemAddr : WordT ; -- memory address
  SIGNAL tData : WordT ; -- memory/io data
  
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 177

```

ALIAS Z : STD_LOGIC IS CR(0); -- Condition code Zero
ALIAS N : STD_LOGIC IS CR(1); -- Condition code Negative
ALIAS C : STD_LOGIC IS CR(2); -- Condition code Carry
ALIAS O : STD_LOGIC IS CR(3); -- Condition code Overflow
ALIAS Opcode : STD_LOGIC_VECTOR(5 DOWNTO 0) IS IR(31 DOWNTO 26);
ALIAS RT : STD_LOGIC_VECTOR(4 DOWNTO 0) IS IR(25 DOWNTO 21);
ALIAS RA : STD_LOGIC_VECTOR(4 DOWNTO 0) IS IR(20 DOWNTO 16);
ALIAS RB : STD_LOGIC_VECTOR(4 DOWNTO 0) IS IR(15 DOWNTO 11);
ALIAS RS : STD_LOGIC_VECTOR(4 DOWNTO 0) IS IR(15 DOWNTO 11);
ALIAS Imm : STD_LOGIC_VECTOR(15 DOWNTO 0) IS IR(15 DOWNTO 0);
ALIAS D : STD_LOGIC_VECTOR(15 DOWNTO 0) IS IR(15 DOWNTO 0);
ALIAS PN : STD_LOGIC_VECTOR(10 DOWNTO 0) IS IR(10 DOWNTO 0);
ALIAS length : STD_LOGIC IS IR(26);
  
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 178

```

  -- other declarations
  CONSTANT delay : TIME := 200 ps; -- register delay
  CONSTANT Reset_delay : TIME := 5 ns;
  CONSTANT Exec_delay : TIME := 10 ns; -- Execute delay
  CONSTANT Mdelay : TIME := 600 ps; -- MemEnable signal delay
  CONSTANT Pulse_Width : TIME := 2.6 ns; -- memory signals width
  CONSTANT Fetch_delay : TIME := 3 ns; -- disable memory after access completed
  
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 179

```

BEGIN
  PROCESS -- transition function
  -- working variables
  VARIABLE RS_data, RA_data, RB_data : WordT;
  VARIABLE RT_addr, RA_addr, RB_addr, RS_addr : Natural;
  BEGIN
    WAIT ON Clk,Reset;
    IF (Reset'Event AND Reset = '1') THEN -- reset function
      PC <= (OTHERS => '0'); CR <= "0000"; IR <= (OTHERS => '0');
      FOR i IN 0 TO 31 LOOP
        GPR(i) <= (OTHERS => '0');
      END LOOP;
      Phase <= p_reset;
      Status <= p_reset;
      WAIT UNTIL (Reset = '0') AND (Clk = '1');
    END IF;
  
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 180

```

IF (ClkEvent AND Clk='1') THEN           -- Instruction cycle
  Status <= Fetch AFTER delay;
  Phase <= Fetch AFTER delay;
  PC <= PC + 4 AFTER Exec_delay;           -- instruction fetch
  WAIT UNTIL MemRdy='1';                 -- wait instr. fetch completed
  IR <= MemData;
  WAIT FOR Fetch_delay;
  Status <= Execute;                       -- instruction execution
  Phase <= Execute;
  RA_addr := CONV_INTEGER('0' & RA); RB_addr := CONV_INTEGER('0' & RB);
  -- '0' to force bit-vector to positive value
  RA_data := GPR(RA_addr); RB_data := GPR(RB_addr);
  RT_addr := CONV_INTEGER('0' & RT);
  RS_addr := CONV_INTEGER('0' & RS); -- source reg. for store
  RS_data := GPR(RS_addr);             -- or I/O write
  WAIT FOR Exec_delay;
  
```

03\_II progetto di sistemi a livello RT

A.A. 2006/07

181

```

CASE Opcode IS
WHEN "000000" => null;                    -- nop
WHEN "000010" => GPR(RT_Addr) <= not (RA_data); -- not
WHEN "000100" => GPR(RT_Addr) <= RA_data(30 DOWNTO 0) & '0'; -- lshift
WHEN "000110" => GPR(RT_Addr) <= '0' & RA_data(31 DOWNTO 1); -- rshift
  -- lrotate
WHEN "001000" => GPR(RT_Addr) <= RA_data(30 DOWNTO 0) & RA_data(31);
  -- rrotate
WHEN "001010" => GPR(RT_Addr) <= RA_DATA(0) & RA_data(31 DOWNTO 1);
WHEN "010000" => GPR(RT_Addr) <= RA_data + RB_data; -- add
WHEN "010001" => GPR(RT_Addr) <= RA_data + Imm;
WHEN "010010" => GPR(RT_Addr) <= RA_data - RB_data; -- sub
WHEN "010011" => GPR(RT_Addr) <= RA_data - Imm;
WHEN "010100" => GPR(RT_Addr) <= RA_data and RB_data; -- and
WHEN "010101" => GPR(RT_Addr) <= RA_data and xt(Imm,RA_data'LENGTH);
  -- ext: zero extension from ieee pkg
  
```

03\_II progetto di sistemi a livello RT

A.A. 2006/07

182

```

WHEN "010110" => GPR(RT_Addr) <= RA_data or RB_data; -- or
WHEN "010111" => GPR(RT_Addr) <= RA_data or ext(Imm,RA_data'LENGTH);
WHEN "011000" => GPR(RT_Addr) <= RA_data xor RB_data; -- xor
WHEN "011001" => GPR(RT_Addr) <= RA_data xor ext(Imm,RA_data'LENGTH);
WHEN "100000" | "100001" =>                -- ldb, ldw
  Phase <= MemOp;
  Status <= MemOp;
  tMemAddr <= RA_data + D;                 -- mem.addr.
  WAIT until MemRdy = '1';
WHEN "100010" | "100011" =>                -- stb, stw
  Phase <= MemOp;
  Status <= MemOp;
  tMemAddr <= RA_data + D;                 -- mem. addr.
  tData <= RS_data; -- mem. data
  WAIT until MemRdy = '1';
  
```

03\_II progetto di sistemi a livello RT

A.A. 2006/07

183

```

WHEN "100100" | "100101" =>                -- irb, irw
  Phase <= IOOp;
  Status <= IOOp;
  WAIT until IORdy = '1';
WHEN "100110" | "100111" =>                -- iwb, iww
  Phase <= IOOp;
  Status <= IOOp;
  tData <= RS_data;                       -- io data
  WAIT until IORdy = '1';
WHEN "110000" => PC <= PC + D;             -- branch
WHEN "110001" => PC <= RA_data(23 DOWNTO 0); -- br.ind.
WHEN "110000" | "110001"
  => IF (N = Opcode(0)) THEN             -- br on N
    PC <= PC + D;
  END IF;
  
```

03\_II progetto di sistemi a livello RT

A.A. 2006/07

184

```

WHEN "110010" | "110011"
  => IF (Z = Opcode(0)) THEN             -- br on Z
    PC <= PC + D;
  END IF;
WHEN "110100" | "110101"
  => IF (C = Opcode(0)) THEN             -- br on C
    PC <= PC + D;
  END IF;
WHEN "110110" | "110111"
  => IF (O = Opcode(0)) THEN             -- br on V
    PC <= PC + D;
  END IF;
WHEN others => null;
END CASE;
  
```

03\_II progetto di sistemi a livello RT

A.A. 2006/07

185

```

IF ((Opcode(5 DOWNTO 4) = 0) or (Opcode(5 DOWNTO 4) = 1))
  and (Opcode /= 0) THEN
  -- set condition register
  IF (GPR(RT_Addr) = 0) THEN CR(0) <= '1'; -- zero result
  ELSE CR(0) <= '0';
  END IF;
  IF (GPR(RT_Addr)(31) = '1') THEN CR(1) <= '1'; -- negative result
  ELSE CR(1) <= '0';
  END IF;
  -- check if operation Opcode generates carry out
  CR(2) <= get_carry(RA_Data, RB_Data, Imm, Opcode);
  -- check if operation Opcode generates overflow
  CR(3) <= get_ovf(RA_Data, RB_Data, Imm, Opcode);
  END IF;
  WAIT FOR 0 ns; -- force signals to be updated
  
```

03\_II progetto di sistemi a livello RT

A.A. 2006/07

186

```

IF (Phase = MemOp) THEN
  IF (dlength = '1') THEN                                -- ldw
    GPR(RT_addr) <= MemData;
  ELSE                                                    -- ldb
    GPR(RT_addr)( 7 DOWNTO 0) <= MemData(7 DOWNTO 0);
    GPR(RT_addr)(31 DOWNTO 8) <= (OTHERS => '0');
  END IF;
  WAIT FOR Fetch_delay;
END IF;
IF (Phase = IOOp) THEN
  IF (dlength = '1') THEN                                -- irw
    GPR(RT_addr) <= IOData;
  ELSE                                                    -- irb
    GPR(RT_addr)( 7 DOWNTO 0) <= IOData(7 DOWNTO 0);
    GPR(RT_addr)(31 DOWNTO 8) <= (OTHERS => '0');
  END IF;
  WAIT FOR Fetch_delay;
END IF;
END PROCESS;

```

03\_II progetto di sistemi a livello RT

```

PROCESS;
BEGIN
  -- output function
  -- Instruction cycle
  WAIT ON Phase;
  IF (Phase = p_reset) THEN                                -- reset
    MemRd <= '0'; MemWr <= '0'; MemEnable <= '0'; MemLength <= '0';
    MemData <= (OTHERS => 'Z');
    IORd <= '0'; IOWr <= '0'; IOEnable <= '0'; IOLength <= '0';
    IOData <= (OTHERS => 'Z');
  ELSIF (Phase = Fetch) THEN                                -- instruction fetch
    MemAddr <= PC AFTER delay;
    MemEnable <= '1' AFTER delay;
    MemRd <= '1' AFTER Mdelay, '0' AFTER Pulse_Width;
    MemLength <= '1' AFTER delay;
    WAIT UNTIL MemRdy='1';                                -- wait instr. fetch completed
    MemEnable <= '0' AFTER Fetch_delay;
  END IF;

```

03\_II progetto di sistemi a livello RT

```

ELSIF (Phase = Execute) THEN NULL;                        -- instruction execution
  -- no output signals
ELSIF (Phase = MemOp) THEN
  MemAddr <= tMemAddr(23 DOWNTO 0) AFTER delay;
  MemEnable <= '1' AFTER delay;
  MemLength <= dlength AFTER delay;
  IF ((To_Bitvector(Opcod) = "100000") OR
    (To_Bitvector(Opcod) = "100001")) THEN                -- ldb, ldw
    MemRd <= '1' AFTER Mdelay, '0' AFTER Pulse_Width;
    WAIT until MemRdy = '1';
    MemEnable <= '0' AFTER Fetch_delay;
    WAIT FOR Fetch_delay;
  END IF;

```

03\_II progetto di sistemi a livello RT

```

IF ((To_Bitvector(Opcod) = "100010") OR
  (To_Bitvector(Opcod) = "100011")) THEN                -- stb, stw
  MemWr <= '1' AFTER Mdelay, '0' AFTER Pulse_Width;
  IF (dlength = '1') THEN                                -- stw
    MemData <= tData AFTER delay;
  ELSE -- stb
    MemData(7 DOWNTO 0) <= tData(7 DOWNTO 0) AFTER delay;
  END IF;
  WAIT until MemRdy = '1';
  MemEnable <= '0' AFTER delay;
  MemData <= (OTHERS => 'Z') AFTER delay;
  WAIT FOR delay;
END IF;

```

03\_II progetto di sistemi a livello RT

```

ELSIF (Phase = IOOp) THEN
  IOAddr <= PN AFTER delay;
  IOEnable <= '1' AFTER delay;
  IOLength <= dlength AFTER delay;
  IF ((To_Bitvector(Opcod) = "100100") OR
    (To_Bitvector(Opcod) = "100101")) THEN                -- irb, irw
    IORd <= '1' AFTER Mdelay, '0' AFTER Pulse_Width;
    WAIT until IORdy = '1';
    IOEnable <= '0' AFTER Fetch_delay;
    WAIT FOR Fetch_delay;
  END IF;

```

03\_II progetto di sistemi a livello RT

```

IF ((To_Bitvector(Opcod) = "100110") OR
  (To_Bitvector(Opcod) = "100111")) THEN                -- iwb, iww
  IF (dlength = '1') THEN                                -- iww
    IOData <= tData AFTER delay;
  ELSE -- iwb
    IOData(7 DOWNTO 0) <= tData(7 DOWNTO 0) AFTER delay;
  END IF;
  IOWr <= '1' AFTER Mdelay, '0' AFTER Pulse_Width;
  WAIT until IORdy = '1';
  IOEnable <= '0' AFTER delay;
  IOData <= (OTHERS => 'Z') AFTER delay;
  WAIT FOR delay;
END IF;
END PROCESS;
END behavioral;

```

03\_II progetto di sistemi a livello RT

## Implementazione

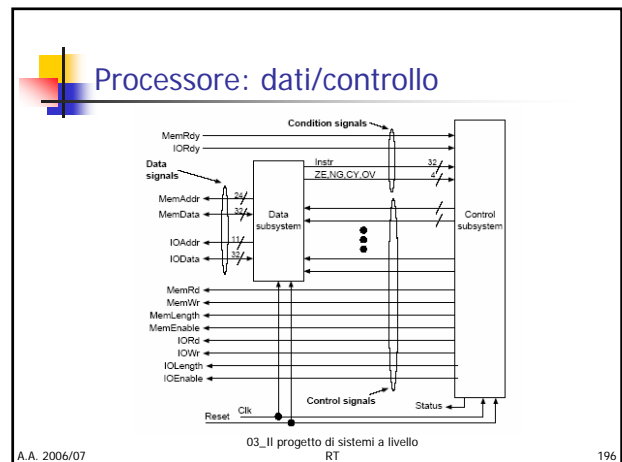
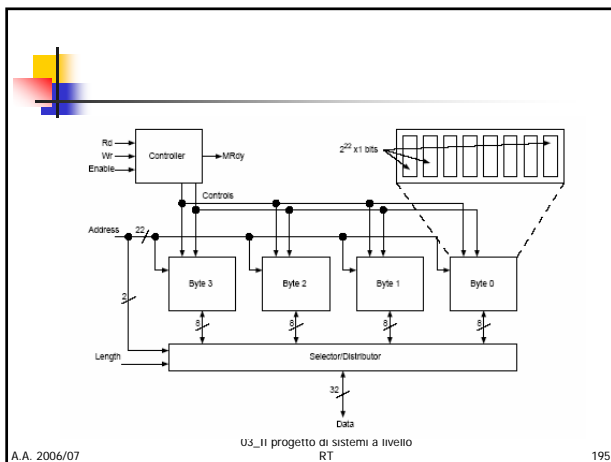
- sottosistema memoria
- processore:
  - sottosistema dati
  - sottosistema controllo

A.A. 2006/07 03\_II progetto di sistemi a livello RT 193

## Sottosistema memoria

- selettore/distributore
- controllore
- chip di memoria ( $2^{22} = 4\text{Mbit}$ , 1 bit in uscita)
- 32 moduli paralleli (128 Mbit = 16 MB)
- output 32 bit o 1 byte alla volta

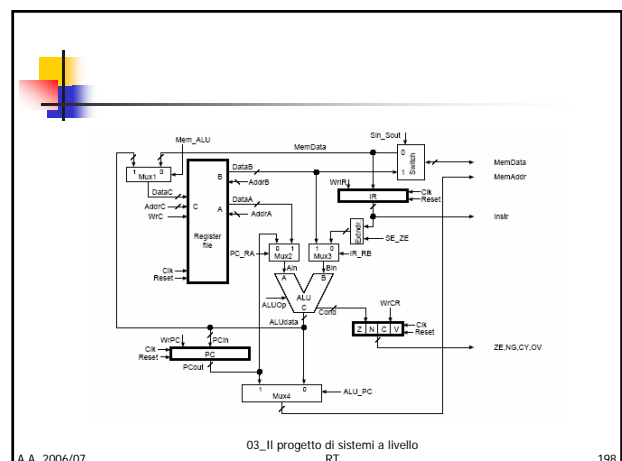
A.A. 2006/07 03\_II progetto di sistemi a livello RT 194



## Sottosistema dati

- general-purpose register file: 32 registri da 32 bit, 2 porte in lettura A e B, 1 in scrittura C. Ogni porta ha il suo indirizzo su 5 bit e i suoi segnali di controllo. 2 letture e 1 scrittura possibili in contemporanea
- registri dedicati per IR (32 bit), PC (32 bit di cui gli 8 MSB a 0) e CR (4 bit) con segnali di caricamento e reset indipendenti
- altra logica di supporto

A.A. 2006/07 03\_II progetto di sistemi a livello RT 197



■ ALU

ALUop	Operation
0000	Zero_32
0001	A + B
0010	A - B
0011	-B
0100	A and B
0101	A or B
0110	A xor B
0111	not(B)
1000	unused
1001	B
1010	shiftl(A)
1011	shiftr(A)
1100	rotl(A)
1101	rotr(A)
1110	A + 4
1111	unused

A.A. 2006/07 03\_II progetto di sistemi a livello RT 199

■ altra logica di supporto:

- MPX
- zero(/sign extender)
- switch

A.A. 2006/07 03\_II progetto di sistemi a livello RT 200

Interconnessioni:

- con la memoria: data bus e address bus
- la porta in scrittura del register file è condivisa tra ALU e memoria
- la porta in lettura B del register file è condivisa tra memoria e porta B della ALU
- l'input A della ALU è condiviso tra register file e PC
- l'input B della ALU è condiviso tra register file e IR

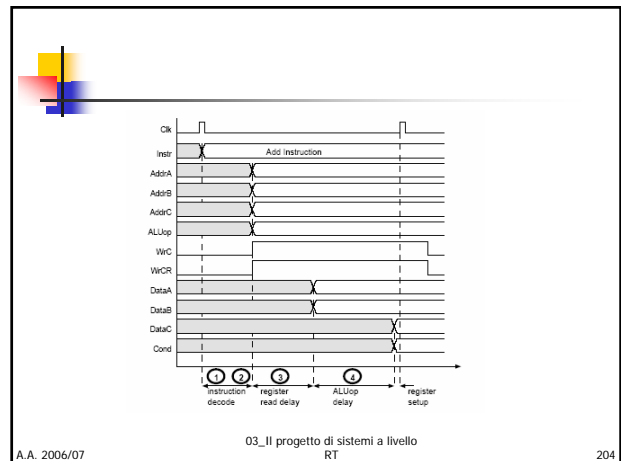
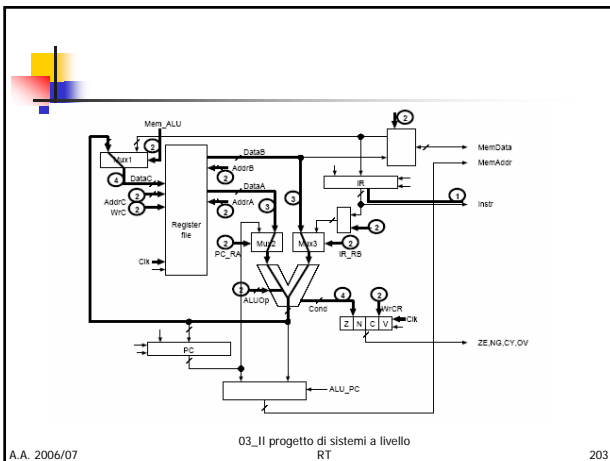
A.A. 2006/07 03\_II progetto di sistemi a livello RT 201

Eventi e flusso dati in esecuzione

Esecuzione di un'istruzione logico-aritmetica:

- Instr (uscita di IR) al sottosistema controllo per decodifica
- dopo la decodifica, il sottosistema controllo attiva i segnali di controllo del register file, della ALU, dei MPX, di memorizzazione dei risultati nei registri
- dopo il ritardo di lettura, sono disponibili come operandi i contenuti dei registri, inviati alla ALU
- dopo il ritardo della ALU, i risultati sono memorizzati nel register file e si setta il CR

A.A. 2006/07 03\_II progetto di sistemi a livello RT 202



### Sottosistema controllo

Rete sequenziale sincrona classica che produce i segnali di controllo sulla base dell'istruzione e dello stato. Codifica one-hot.

A.A. 2006/07 03\_II progetto di sistemi a livello RT 205

### Diagramma di stato

- Fetch: l'istruzione è letta dalla memoria in IR. L'ALU calcola l'indirizzo della prossima istruzione sequenziale e lo memorizza in PC
- Execute: l'istruzione viene decodificata, si reggono gli operandi dai regisri, l'ALU esegue su questi l'operazione, si memorizza il risultato in un registro. Per istruzioni di memoria e di salto si calcola l'indirizzo effettivo e lo si memorizza in PC
- Memop: esecuzione di letture/scritture in memoria

A.A. 2006/07 03\_II progetto di sistemi a livello RT 206

A.A. 2006/07 03\_II progetto di sistemi a livello RT 207

### Descrizione dell'implementazione

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE WORK.comp_pkg.ALL, WORK.ALL;

ENTITY Data_Subsystem IS
    PORT(MemAddr : OUT MAddrT;
         MemData : INOUT WordT;
         IOAddr : OUT IOAddrT;
         IOData : INOUT WordT;
         Instr : OUT WordT;
         ZE, NG, CY, OV : OUT STD_LOGIC;
         AddrA, AddrB, AddrC : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
         ALUOp : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         WrC, WrPC, WrCR, WriR : IN STD_LOGIC;
         Mem_ALU, PC_RA, IR_RB : IN STD_LOGIC;
         ALU_PC, ZE_SE, Sin_Sout : IN STD_LOGIC);
END Data_Subsystem;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 208

```

BEGIN
P1: ENTITY Data_Subsystem
    PORT MAP (MemAddr, MemData, IOAddr, IOData,
             Instr, ZE, NG, CY, OV, AddrA, AddrB, AddrC, ALUOp,
             WrC, WrPC, WrCR, WriR, Mem_ALU, PC_RA, IR_RB,
             ALU_PC, ZE_SE, Sin_Sout, Clk, Reset);
P2: ENTITY Ctrl_Subsystem
    PORT MAP (Instr, ZE, NG, CY, OV, AddrA, AddrB, AddrC, ALUOp,
             WrC, WrPC, WrCR, WriR, Mem_ALU, PC_RA, IR_RB,
             ALU_PC, ZE_SE, Sin_Sout, MemRd, MemWr, MemLength,
             MemEnable, MemRdy, IORd, IOWr, IOLength, IOEnable,
             IORdy, Status, Clk, Reset);
END structural;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 209

### Sottosistema dati

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE WORK.comp_pkg.ALL, WORK.ALL;

ENTITY Data_Subsystem IS
    PORT(MemAddr : OUT MAddrT;
         MemData : INOUT WordT;
         IOAddr : OUT IOAddrT;
         IOData : INOUT WordT;
         Instr : OUT WordT;
         ZE, NG, CY, OV : OUT STD_LOGIC;
         AddrA, AddrB, AddrC : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
         ALUOp : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         WrC, WrPC, WrCR, WriR : IN STD_LOGIC;
         Mem_ALU, PC_RA, IR_RB : IN STD_LOGIC;
         ALU_PC, ZE_SE, Sin_Sout : IN STD_LOGIC);
END Data_Subsystem;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 210

```

ARCHITECTURE structural OF Data_Subsystem IS
SIGNAL DataA, DataB, DataC : WordT ;
SIGNAL Ain , Bin : WordT ;
SIGNAL ALUdata, IRdata : WordT ;
SIGNAL tMemdata : WordT ;
SIGNAL Cond, CRout : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL IRreg, IRExt : WordT ;
SIGNAL PCout : WordT:= (OTHERS => '0');
BEGIN
  ALU1: ENTITY ALU
    PORT MAP(Ain,Bin,ALUop,ALUdata,Cond);
  GPR: ENTITY Reg_File
    PORT MAP(AddrA,AddrB,AddrC,DataA,DataB,DataC, WrC,Reset,Clk);
  PC: ENTITY Reg
    PORT MAP(ALUdata(23 DOWNTO 0),PCout(23 DOWNTO 0),
              rPC,Reset,Clk);
  MX1: ENTITY Mux
    PORT MAP(tMemData,ALUdata,Mem_ALU_DataC);

```

A.A. 2006/07

03\_II progetto di sistemi a livello RT

211

```

CR: ENTITY Reg
  PORT MAP(Cond,CRout,WrCR,Reset,Clk);
  ZE <= CRout(0); CY <= CRout(1);
  NG <= CRout(2); OV <= CRout(3);
IR: ENTITY Reg
  PORT MAP(tMemData,IRReg,WrIR,Reset,Clk);
  Instr <= IRReg;
MX2: ENTITY Mux
  PORT MAP(PCout,DataA,PC_RA,Ain);
ZSE: ENTITY Extender
  PORT MAP(IRreg,ZE_SE,IRExt);
MX3: ENTITY Mux
  PORT MAP(IRExt,DataB,IR_RB,Bin);
MX4: ENTITY Mux
  PORT MAP(ALUdata(23 DOWNTO 0),PCout(23 DOWNTO 0),
            ALU_PC,MemAddr);
SL: ENTITY Switch
  PORT MAP(MemData,tMemData,DataB,Sin_Sout);
END structural;

```

A.A. 2006/07

03\_II progetto di sistemi a livello RT

212

**Register file**

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE WORK.comp_pkg.ALL;

ENTITY Reg_File IS
  PORT(AddrA, AddrB, AddrC : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
        DataA, DataB : OUT WordT;
        DataC : IN WordT;
        WrC : IN STD_LOGIC ;
        Reset, Clk : IN STD_LOGIC);
END Reg_File;

```

A.A. 2006/07

03\_II progetto di sistemi a livello RT

213

```

ARCHITECTURE behavioral OF Reg_File IS
  TYPE RegFileT IS ARRAY(0 to 31) OF WordT;
  SIGNAL GPR : RegFileT ;
BEGIN
  PROCESS(AddrA,AddrB)
    CONSTANT RF_delay : TIME := 4 ns;
    DATA A <= GPR(CONV_INTEGER(AddrA)) AFTER RF_delay;
    DATA B <= GPR(CONV_INTEGER(AddrB)) AFTER RF_delay;
  END PROCESS;

```

A.A. 2006/07

03\_II progetto di sistemi a livello RT

214

```

PROCESS(Reset,Clk) -- transition function
BEGIN
  IF (Reset'EVENT and (Reset = '1')) THEN
    FOR i IN 0 TO 31 LOOP
      GPR(i) <= (OTHERS => '0');
    END LOOP;
  END IF;
  IF (Clk'EVENT AND Clk = '1' AND WrC = '1') THEN
    GPR(CONV_INTEGER(AddrC)) <= DataC;
  END IF;
END PROCESS;
END behavioral;

```

A.A. 2006/07

03\_II progetto di sistemi a livello RT

215

**ALU**

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;
USE WORK.comp_pkg.ALL;

ENTITY ALU IS
  PORT(A, B : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        Op : IN STD_LOGIC_VECTOR( 3 DOWNTO 0);
        C : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        Cond : OUT STD_LOGIC_VECTOR( 3 DOWNTO 0));
END ALU;

```

A.A. 2006/07

03\_II progetto di sistemi a livello RT

216

**ARCHITECTURE behavioral OF ALU IS**  
**BEGIN**  
**PROCESS(A,B,Op)**  
**CONSTANT** ALU\_delay : **TIME** := 6 ns;  
**BEGIN**  
**CASE Op IS**  
**WHEN** "0000" => C <= (OTHERS => '0') **AFTER** ALU\_delay;  
**WHEN** "0001" => C <= A + B **AFTER** ALU\_delay;  
**WHEN** "0010" => C <= A - B **AFTER** ALU\_delay;  
**WHEN** "0011" => C <= (OTHERS => '0') **AFTER** ALU\_delay;  
**WHEN** "0100" => C <= A and B **AFTER** ALU\_delay;  
**WHEN** "0101" => C <= A or B **AFTER** ALU\_delay;  
**WHEN** "0110" => C <= A xor B **AFTER** ALU\_delay;  
**WHEN** "0111" => C <= (OTHERS => '0') **AFTER** ALU\_delay;

A.A. 2006/07 03\_II progetto di sistemi a livello RT 217

**WHEN** "1000" => C <= A **AFTER** ALU\_delay;  
**WHEN** "1001" => C <= B **AFTER** ALU\_delay;  
**WHEN** "1010" => C <= A(30 **DOWNTO** 0) & '0' **AFTER** ALU\_delay;  
**WHEN** "1011" => C <= '0' & A(31 **DOWNTO** 1) **AFTER** ALU\_delay;  
**WHEN** "1100" => C <= A(30 **DOWNTO** 0) & A(31) **AFTER** ALU\_delay;  
**WHEN** "1101" => C <= A(0) & A(31 **DOWNTO** 1) **AFTER** ALU\_delay;  
**WHEN** "1110" => C <= A + 4 **AFTER** ALU\_delay;  
**WHEN** "1111" => C <= not(A) **AFTER** ALU\_delay;  
**WHEN OTHERS** => NULL;  
**END CASE**;  
**Cond** <= get\_cc(A,B,Op) **AFTER** ALU\_delay;  
**END PROCESS**;  
**END behavioral**;

A.A. 2006/07 03\_II progetto di sistemi a livello RT 218

**Registri**

**LIBRARY** ieee; **USE** ieee.std\_logic\_1164.ALL;  
**ENTITY** Reg **IS**  
**PORT**(Data\_in : **IN** STD\_LOGIC\_VECTOR;  
Data\_out: **OUT** STD\_LOGIC\_VECTOR;  
Wr : **IN** STD\_LOGIC ;  
Reset : **IN** STD\_LOGIC ;  
Clk : **IN** STD\_LOGIC);  
**END** Reg;

A.A. 2006/07 03\_II progetto di sistemi a livello RT 219

**ARCHITECTURE behavioral OF Reg IS**  
**BEGIN**  
**PROCESS**(Wr,Reset,Clk)  
**CONSTANT** Reg\_delay: **TIME** := 2 ns;  
**VARIABLE** BVZero: STD\_LOGIC\_VECTOR(Data\_in **RANGE**):=  
(OTHERS => '0');  
**BEGIN**  
**IF** (Reset = '1') **THEN**  
Data\_out <= BVZero **AFTER** Reg\_delay;  
**END IF**;  
**IF** (Clk'EVENT AND Clk = '1' AND Wr = '1') **THEN**  
Data\_out <= Data\_in **AFTER** Reg\_delay;  
**END IF**;  
**END PROCESS**;  
**END behavioral**;

A.A. 2006/07 03\_II progetto di sistemi a livello RT 220

**Altri moduli: mux**

**LIBRARY** ieee;  
**USE** ieee.std\_logic\_1164.ALL;  
**ENTITY** Mux **IS**  
**PORT**(A\_in,B\_in: **IN** STD\_LOGIC\_VECTOR;  
Sel : **IN** STD\_LOGIC ;  
Data\_out : **OUT** STD\_LOGIC\_VECTOR);  
**END** Mux;  
**ARCHITECTURE behavioral OF Mux IS**  
**BEGIN**  
**PROCESS**(A\_in, B\_in, Sel)  
**CONSTANT** Mux\_delay: **TIME** := 500 ps;  
**BEGIN**  
**IF** (Sel = '0') **THEN**  
Data\_out <= A\_in **AFTER** Mux\_delay;  
**ELSE**  
Data\_out <= B\_in **AFTER** Mux\_delay;  
**END IF**;  
**END PROCESS**;  
**END behavioral**;

A.A. 2006/07 03\_II progetto di sistemi a livello RT 221

**Altri moduli: extender**

**LIBRARY** ieee;  
**USE** ieee.std\_logic\_1164.ALL;  
**ENTITY** Extender **IS**  
**PORT**(X\_in : **IN** STD\_LOGIC\_VECTOR(31 **DOWNTO** 0);  
ZE\_SE : **IN** STD\_LOGIC ;  
X\_out : **OUT** STD\_LOGIC\_VECTOR(31 **DOWNTO** 0));  
**END** Extender;

A.A. 2006/07 03\_II progetto di sistemi a livello RT 222

```

ARCHITECTURE behavioral OF Extender IS
BEGIN
PROCESS(X_in, ZE_SE)
CONSTANT Ext_delay: TIME := 500 ps;
BEGIN
IF (ZE_SE = '0') THEN
X_out(31 DOWNT0 16) <= (OTHERS => '0') AFTER Ext_delay;
X_out(15 DOWNT0 0) <= X_in(15 DOWNT0 0) AFTER Ext_delay;
ELSE
X_out(31 DOWNT0 16) <= (OTHERS => X_in(15)) AFTER Ext_delay;
X_out(15 DOWNT0 0) <= X_in(15 DOWNT0 0) AFTER Ext_delay;
END IF;
END PROCESS;
END behavioral;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 223

Altri moduli: switch

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Switch IS
PORT(A : INOUT STD_LOGIC_VECTOR;
      B_out: OUT STD_LOGIC_VECTOR;
      C_in : IN STD_LOGIC_VECTOR;
      Sel : IN STD_LOGIC );
END Switch;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 224

```

ARCHITECTURE behavioral OF Switch IS
BEGIN
PROCESS(A, C_in, Sel)
CONSTANT Switch_delay: TIME := 500 ps;
CONSTANT dataZ: STD_LOGIC_VECTOR(A'RANGE):= (OTHERS => 'Z');
BEGIN
IF (Sel = '0') THEN
B_out <= A AFTER Switch_delay;
A <= dataZ;
ELSE
A <= C_in AFTER Switch_delay;
END IF;
END PROCESS;
END behavioral;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 225

Sottosistema controllo

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE WORK.comp_pkg.ALL;
ENTITY Ctrl_Subsystem IS
PORT(Instr : IN WordT ; ZE, NG, CY, OV : IN STD_LOGIC ;
      AddrA, AddrB, AddrC : OUT STD_LOGIC_VECTOR(4 DOWNT0 0);
      ALUOp : OUT STD_LOGIC_VECTOR(3 DOWNT0 0);
      WrC, WrPC, WrCR, WrIR : OUT STD_LOGIC ;
      Mem_ALU, PC_RA, IR_RB : OUT STD_LOGIC ;
      ALU_PC, ZE_SE, Sin_Sout: OUT STD_LOGIC ;
      MemRd, MemWr : OUT STD_LOGIC; MemLength : OUT STD_LOGIC;
      MemEnable : OUT STD_LOGIC ; MemRdy : IN STD_LOGIC ;
      IORd, IOWr : OUT STD_LOGIC ; IOLength : OUT STD_LOGIC ;
      IOEnable : OUT STD_LOGIC ; IORdy : IN STD_LOGIC ;
      Status : OUT StatusT ; Clk, Reset : IN STD_LOGIC );
END Ctrl_Subsystem;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 226

```

LIBRARY ieee;
USE ieee.std_logic_signed.ALL;
ARCHITECTURE behavioral OF Ctrl_Subsystem IS
SIGNAL State: StatusT;
BEGIN
PROCESS -- transition function
ALIAS Opcode : STD_LOGIC_VECTOR(5 DOWNT0 0) IS Instr(31 DOWNT0 26);
CONSTANT Reset_delay: TIME:= 500 ps ;
CONSTANT Ctrl_delay : TIME:= 500 ps ;
BEGIN
WAIT ON Clk, Reset;
IF (Reset'EVENT AND Reset = '1') THEN
State <= p_reset AFTER Reset_delay;
Status <= p_reset AFTER Reset_delay;
WAIT UNTIL Clk = '1';
END IF;
IF (Clk'EVENT) AND (Clk = '1') THEN
CASE State IS
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 227

```

WHEN p_reset => Status <= fetch AFTER Ctrl_delay;
State <= fetch AFTER Ctrl_delay;
WHEN fetch => Status <= execute AFTER Ctrl_delay;
State <= execute AFTER Ctrl_delay;
WHEN execute => CASE Opcode IS
WHEN "100000" | "100001" => State <= memop AFTER Ctrl_delay;
Status <= memop AFTER Ctrl_delay;
WHEN "100010" | "100011" => State <= memop AFTER Ctrl_delay;
Status <= memop AFTER Ctrl_delay;
WHEN OTHERS => State <= fetch AFTER Ctrl_delay;
Status <= fetch AFTER Ctrl_delay;
END CASE;
WHEN memop | ioop => Status <= fetch AFTER Ctrl_delay;
State <= fetch AFTER Ctrl_delay;
WHEN undef => NULL;
END CASE;
END IF;
END PROCESS;
    
```

A.A. 2006/07 03\_II progetto di sistemi a livello RT 228

```

PROCESS(State,Instr,MemRdy) -- output function
ALIAS Opcode : STD_LOGIC_VECTOR( 5 DOWNT0 0) IS Instr(31 DOWNT0 26);
ALIAS Imm : STD_LOGIC_VECTOR(15 DOWNT0 0) IS Instr(15 DOWNT0 0);
ALIAS D : STD_LOGIC_VECTOR(15 DOWNT0 0) IS Instr(15 DOWNT0 0);
ALIAS PN : STD_LOGIC_VECTOR(10 DOWNT0 0) IS Instr(10 DOWNT0 0);
CONSTANT Dec_delay : TIME:= 3 ns;
CONSTANT Ctrl_delay : TIME:= 500 ps;
CONSTANT MemRd_delay : TIME:= 2500 ps;
CONSTANT MemRd_pulse : TIME:= MemRd_delay + 3 ns ;
CONSTANT MemWr_delay : TIME:= 2500 ps;
CONSTANT MemWr_pulse : TIME:= MemWr_delay + 3 ns ;
TYPE Ctrl_LineT IS
RECORD
    MemOp, WrMem : STD_LOGIC;
    RS_RB, IR_RB : STD_LOGIC;
    WrC, WrPC, WrCR : STD_LOGIC;
    ZE_SE : STD_LOGIC;
    ALUop : STD_LOGIC_VECTOR(3 DOWNT0 0);
END RECORD;
03_ Il progetto di sistemi a livello
A.A. 2006/07 RT 229
    
```

```

VARIABLE Ctrl_Line : Ctrl_LineT;
TYPE Ctrl_TableT IS ARRAY(NATURAL RANGE 0 TO 63) OF Ctrl_LineT;
CONSTANT Ctrl_Table : Ctrl_TableT:=
-- Mem Wr RS IR Wr Wr Wr Wr ZE ALU
-- Op Mem RB RB C PC CR SE op
(0 => ('0', '0', '1', '1', '0', '0', '0', '0', '0', '0000'), -- nop
2 => ('0', '0', '1', '1', '1', '1', '0', '1', '0', '1111'), -- not
4 => ('0', '0', '1', '1', '1', '0', '1', '0', '0', '1010'), -- lsh
6 => ('0', '0', '1', '1', '1', '1', '0', '1', '0', '1011'), -- rsh
8 => ('0', '0', '1', '1', '1', '0', '1', '0', '0', '1100'), -- lrt
10=> ('0', '0', '1', '1', '1', '1', '0', '1', '0', '1101'), -- rrt
16=> ('0', '0', '1', '1', '1', '0', '1', '0', '0', '0001'), -- add
17=> ('0', '0', '1', '0', '1', '0', '1', '1', '1', '0001'), -- adi
18=> ('0', '0', '1', '1', '1', '0', '1', '1', '0', '0010'), -- sub
19=> ('0', '0', '1', '0', '1', '0', '1', '1', '1', '0010'), -- sbi
20=> ('0', '0', '1', '1', '1', '1', '0', '1', '0', '0100'), -- and
21=> ('0', '0', '1', '0', '1', '1', '0', '1', '0', '0100'), -- ani
03_ Il progetto di sistemi a livello
A.A. 2006/07 RT 230
    
```

```

22=> ('0', '0', '1', '1', '1', '0', '1', '0', '0', '0101'), -- or
23=> ('0', '0', '1', '0', '1', '0', '1', '0', '0', '0101'), -- ori
24=> ('0', '0', '1', '1', '1', '0', '1', '0', '0', '0110'), -- xor
25=> ('0', '0', '1', '0', '1', '0', '1', '0', '0', '0110'), -- xri
32=> ('1', '0', '0', '0', '0', '1', '0', '0', '1', '0001'), -- ldb
33=> ('1', '0', '0', '0', '0', '1', '0', '0', '1', '0001'), -- ldw
34=> ('1', '1', '0', '0', '0', '0', '0', '0', '1', '0001'), -- stb
35=> ('1', '1', '0', '0', '0', '0', '0', '0', '1', '0001'), -- stw
36=> ('1', '0', '0', '1', '1', '0', '0', '0', '0', '1001'), -- irb
37=> ('1', '0', '0', '1', '1', '1', '0', '0', '0', '1001'), -- irw
38=> ('1', '1', '0', '1', '1', '0', '0', '0', '0', '1001'), -- iwb
39=> ('1', '1', '0', '1', '1', '0', '0', '0', '0', '1001'), -- iww
56=> ('0', '0', '1', '0', '0', '1', '0', '0', '1', '0001'), -- br
57=> ('0', '0', '1', '0', '0', '1', '0', '1', '1', '0000'), -- bri
48=> ('0', '0', '1', '0', '0', '1', '0', '1', '0', '0001'), -- brp
49=> ('0', '0', '1', '0', '0', '1', '0', '1', '0', '0001'), -- brn
50=> ('0', '0', '1', '0', '0', '1', '0', '1', '0', '0001'), -- bnz
03_ Il progetto di sistemi a livello
A.A. 2006/07 RT 231
    
```

```

51=> ('0', '0', '1', '0', '0', '1', '0', '1', '0', '0001'), -- brz
52=> ('0', '0', '1', '0', '0', '1', '0', '1', '0', '0001'), -- bnc
53=> ('0', '0', '1', '0', '0', '1', '0', '1', '0', '0001'), -- brc
54=> ('0', '0', '1', '0', '0', '1', '0', '1', '0', '0001'), -- bnv
55=> ('0', '0', '1', '0', '0', '1', '0', '1', '0', '0001'), -- brv
OTHERS => ('0', '0', '1', '1', '0', '0', '0', '1', '0000')
);
BEGIN
IF (State EVENT) THEN
CASE State IS
WHEN undef => NULL;
WHEN p_reset => ALUOp <= "0000";
                MemRd <= '0'; MemWr <= '0';
                MemEnable <= '0'; MemLength <= '0';
                IORd <= '0'; IOWr <= '0';
                IOEnable <= '0'; IOLength <= '0';
03_ Il progetto di sistemi a livello
A.A. 2006/07 RT 232
    
```

```

WHEN fetch =>
-- disable write signals from previous cycle
WrCR <= '0' AFTER Ctrl_delay;
WrC <= '0' AFTER Ctrl_delay;
-- fetch instruction
ALU_PC <= '1' AFTER Ctrl_delay;
MemLength<= '1' AFTER Ctrl_delay;
MemEnable<= '1' AFTER Ctrl_delay;
MemRd <= '1' AFTER MemRd_delay, '0' AFTER MemRd_pulse;
Sin_Sout <= '0' AFTER Ctrl_delay; -- switch in
-- increment PC
PC_RA <= '0' AFTER Ctrl_delay;
ALUop <= "1110" AFTER Ctrl_delay; -- PC + 4
WrIR <= '1' AFTER Ctrl_delay;
WrPC <= '1' AFTER Ctrl_delay;
03_ Il progetto di sistemi a livello
A.A. 2006/07 RT 233
    
```

```

WHEN execute =>
-- disable signals from fetch cycle
WrIR <= '0' AFTER Ctrl_delay;
WrPC <= '0' AFTER Ctrl_delay;
MemEnable<= '0' AFTER Ctrl_delay;
-- other actions done by InstrEVENT
WHEN memop | ioop =>
-- initiate memory access
ALU_PC <= '0' AFTER Ctrl_delay; -- address to memory
MemEnable<= '1' AFTER Ctrl_delay;
MemLength<= Opcode(0) AFTER Ctrl_delay; -- operand length
WrC <= Ctrl_Line.WrC AFTER Ctrl_delay;
03_ Il progetto di sistemi a livello
A.A. 2006/07 RT 234
    
```

```

IF (Ctrl_Line.WrMem = '0') THEN
  MemRd <= '1' AFTER MemRd_delay, '0' AFTER MemRd_pulse;
  Mem_ALU <= '0' AFTER Ctrl_delay;
ELSE
  MemWr <= '1' AFTER MemWr_delay, '0' AFTER MemWr_pulse;
  Sin_Sout <= '1' AFTER Ctrl_delay;
END IF;
END CASE;
END IF;
IF (InstrEVENT) THEN
  -- decode opcode
  Ctrl_Line:= Ctrl_Table(CONV_INTEGER('0' & Opcode));

```

03\_II progetto di sistemi a livello  
RT

A.A. 2006/07 235

```

-- decode registers
AddrA <= Instr(20 DOWNT0 16) AFTER Dec_delay;
IF (Ctrl_Line.RS_RB = '0') THEN
  AddrB <= Instr(25 DOWNT0 21) AFTER Dec_delay;
ELSE
  AddrB <= Instr(15 DOWNT0 11) AFTER Dec_delay;
END IF;
AddrC <= Instr(25 DOWNT0 21) AFTER Dec_delay;
-- decode control signals
PC_RA <= not(Ctrl_Line.WrPC) AFTER Ctrl_delay;
ZE_SE <= Ctrl_Line.ZE_SE AFTER Ctrl_delay;
IR_RB <= Ctrl_Line.IR_RB AFTER Ctrl_delay;
ALUOp <= Ctrl_Line.ALUop AFTER Ctrl_delay;
WrPC <= Ctrl_Line.WrPC AFTER Ctrl_delay;
WrCR <= Ctrl_Line.WrCR AFTER Ctrl_delay;
IF (Ctrl_Line.MemOp = '0') THEN
  WrC <= Ctrl_Line.WrC AFTER Ctrl_delay;
  Mem_ALU <= '1' AFTER Ctrl_delay;
END IF;
END IF;

```

03\_II progetto di sistemi a livello  
RT

A.A. 2006/07 236

```

IF (MemRdyEVENT AND MemRdy='1') THEN
  CASE State IS
    WHEN memop => IF (Ctrl_Line.WrMem = '1') THEN
      -- deactivate data bus
      Sin_Sout <= '0' AFTER Ctrl_delay;
      END IF;
    WHEN OTHERS => NULL;
  END CASE;
END IF;
END PROCESS;
END behavioral;

```

03\_II progetto di sistemi a livello  
RT

A.A. 2006/07 237