

ARCHITETTURA

INDICE

1. Sistema di numerazione posizionale pesato
 - 1.1. Rappresentazione dell'informazione
 - 1.2. Introduzione al sistema posizionale pesato
 - 1.3. Algoritmi di conversione e aritmetica binaria
 - 1.4. Standard modulo e segno (MS)
 - 1.5. Standard complemento a due (C2)
 - 1.6. Standard in virgola mobile
2. Funzioni binarie di variabile binaria
 - 2.1. Funzione binaria di variabile binaria
 - 2.2. Mintermini e maxtermini
 - 2.3. Algebra di commutazione
 - 2.4. Leggi di De Morgan
3. Componenti fondamentali di base
 - 3.1. Encoder (Codificatore)
 - 3.2. Decoder (Decodificatore)
 - 3.3. Multiplexer
 - 3.4. Demultiplexer
 - 3.5. Sommatore
4. Progettazione MIPS a ciclo singolo
 - 4.1. Istruzioni e formati
 - 4.2. Unita logico aritmetica (ALU)
 - 4.3. Flip Flop

- 4.4. Banco dei registri
- 4.5. Memoria Istruzioni / Memoria Dati
- 4.6. Control Unit
- 4.7. Fasi del processore
- 4.8. Introduzione salti
- 4.9. Progetto finale mips ciclo singolo
- 5. Progettazione MIPS con struttura pipeline
 - 5.1. Introduzione
 - 5.1.1. Introduzione alla pipeline
 - 5.1.2. Calcolo delle prestazioni
 - 5.1.3. Criticità strutturali
 - 5.1.4. Criticità sui dati
 - 5.1.5. Criticità sul controllo
 - 5.2. Introduzione dell'unità di propagazione
 - 5.3. Introduzione dell'unità di individuazione del conflitto
 - 5.4. Anticipazione della BEQ
 - 5.5. Progetto finale mips con struttura pipeline
- 6. Gerarchia di memoria
 - 6.1. Prima panoramica
 - 6.2. Criteri di Gestione
 - 6.3. Corrispondenza tra indirizzi in memoria e in cache

1. SISTEMA DI NUMERAZIONE POSIZIONALE PESATO

1.1 RAPPRESENTAZIONE DELL'INFORMAZIONE

L'unità di informazione è per convenzione la quantità di informazione necessaria a dimezzare l'incertezza tra le scelte possibili (BIT).

Per individuare un numero tra 2^n numeri occorrono n bit

1.2 INTRODUZIONE AL SISTEMA POSIZIONALE PESATO

Per prima cosa si sceglie una base (2, ... , 8 , .. 10 , .. 16 ...) e si dà un esponente alla base a seconda della posizione.

Esempio :

$$(2425,295)_{10} = 2 \times 10^3 + 4 \times 10^2 + 2 \times 10^1 + 5 \times 10^0 + 2 \times 10^{-1} + 9 \times 10^{-2} + 5 \times 10^{-3}$$

$$(10110101)_2 = +1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

1.3 ALGORITMI DI CONVERSIONE

Algoritmo di conversione da binario a decimale.

Si parte dal bit più significativo, mettendo caso di avere un numero a 4 bit(1011):

$$S3=b3$$

$$S3=1$$

$$S2= b2+ 2 \cdot S3$$

$$S2=0+2 \cdot 1=2$$

$$S1=b1+ 2 \cdot S2$$

$$S1=1+2 \cdot 2=5$$

$$S0=b0+2 \cdot S1$$

$$S0=1+2 \cdot 5=11$$

Il valore effettivo del numero sarà quello di S0.

Algoritmo di conversione da decimale a binario.

Volendo fare l'esempio inverso(11):

$$S_0 = a_0 + 2 \cdot S_1$$

$$11 = S_0 = 1 + 2 \cdot 5$$

$$S_1 = a_1 + 2 \cdot S_2$$

$$5 = S_1 = 1 + 2 \cdot 2$$

$$S_2 = a_2 + 2 \cdot S_3$$

$$2 = S_2 = 0 + 2 \cdot 1$$

$$S_3 = a_3 + 2 \cdot S_4$$

$$1 = S_3 = 1 + 2 \cdot 0$$

Il numero si legge a partire da $a_3(1011)$ e ci si ferma solo quando $S_{i+1}=0$.

Algoritmo di conversione da ottale a binario.

Una sequenza di numeri scritta in ottale viene tradotta in blocchi da 3 poiché $8=2^3$.

Esempio $(213)_8 = 010 | 001 | 011$ poiché $2=010$, $1=001$, $3=011$.

$(010001011)_2 = 139$.

Algoritmo di conversione da binario a esadecimale.

Prendendo una sequenza di bit scritta in binario devo prenderla in blocchi da 4 poiché $2^4=16$. Esempio $(010001011)_2 = (08b)_{16}$

NOTA: Nel binario puro con n bit fissati posso rappresentare da 0 a 2^n-1 .

1.4 STANDARD MODULO E SEGNO (MS)

Nel binario puro possiamo indicare solamente numeri positivi e nasce quindi la necessità di ampliare la rappresentazione introducendo lo standard Modulo e Segno.

In questa rappresentazione il bit più significativo individua il segno: 1(-), 0(+).

Nel modulo e segno possiamo rappresentare numeri da -2^{n-1} fino a $2^{n-1}-1$. Perdiamo un numero poiché avendo introdotto il segno qualsiasi numero con il bit più significativo 1 e tutti gli altri 0 non risulterà avere valore poiché dovrebbe corrispondere a -0.

ESEMPIO: 100, questo numero non ha valore nella rappresentazione MS.

1.5 STANDARD COMPLEMENTO A DUE (C2)

Per quanto riguarda i numeri positivi il C2 combacia con lo standard MS; per quanto riguarda i numeri negativi invece il C2 risolve il problema della perdita di un

numero e la sua rappresentazione va da -2^n a 2^n-1 .

Per indicare un numero N negativo esso viene scritto in tal maniera:

$$N=2^{n-1} - |\text{MODULO}|.$$

Esempio: -2 su 4(n) bit:

$$2^{4-1} - |2| = 8-2=6=1(-) | 110(6).$$

1.6 STANDARD IN VIRGOLA MOBILE

Lo standard in virgola mobile nasce data la necessità di poter rappresentare in forma normalizzata un non intero che con la notazione esponenziale può essere rappresentato in infiniti modi.

$$\text{ESEMPIO: } 2.34 = 234 \cdot 10^{-2} = 0.0234 \cdot 10^2.$$

È quindi evidente il bisogno di una rappresentazione standardizzata e per quanto riguarda il binario si prende in considerazione la rappresentazione con un solo valore maggiore di zero a sinistra della virgola.

Per la rappresentazione in virgola mobile vengono utilizzati 32 bit, il primo è il bit di segno, i seguenti 8 sono dedicati all'esponente da dare per una base fissata(2) e i restanti 23 bit sono dedicati alla mantissa e cioè alla parte dopo la virgola.

Poiché abbiamo 8 bit possiamo indicare numeri che vanno da 0 a 255, avendo però la necessità di un esponente negativo dividiamo il nostro intervallo da -126 a 127. Se indichiamo con "E" l'esponente che possediamo e con "e" l'esponente da convertire in binario utilizzerò la formula: $E=e-127$.

ESEMPIO, $E=2$, l'esponente da convertire in binario sarà $e=127+2=129$.

I restanti 23 bit indicano ciò che si trova a destra della virgola prendendo come standard la presenza di un solo 1 a sinistra di questa.

ESEMPIO, $(-5, 828125)_{10}$, valgono le regole di conversioni generali per la parte intera e quella frazionaria, il numero in binario sarà quindi $(101,110101)_2$.

Nasce ora la necessità di normalizzarlo portando la virgola alla destra del primo 1---> $(1,01110101)_2 \cdot 10^2$, l'esponente vale +2 poiché devo spostare la virgola di due posizioni verso destra. $E=2$ quindi il nostro esponente sarà 129 che in binario su 8 bit vale 10000001. Abbiamo ora tutto il necessario per rappresentare il numero in virgola mobile su 32 bit: 1 | 10000001 | 011101010000000000000000.

Il primo 1 si omette sempre poiché è l'unico valore possibile

2. FUNZIONI BINARIE DI VARIABILE BINARIA

2.1 FUNZIONE BINARIA DI VARIABILE BINARIA

Una funzione binaria di variabile binaria $y = f(x)$ è una funzione che ha in input un certo numero di bit e restituisce in output un unico bit di informazione secondo una vera e propria funzione matematica.

Esempio delle possibili funzioni binarie di variabile binaria che ricevono in input 1 bit

Input	Output			
x_0	y_0	y_1	y_2	y_3
0	0	0	1	1
1	0	1	0	1

La funzione y_2 è detta funzione **NOT**.

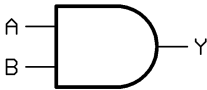
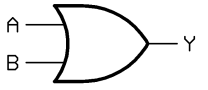
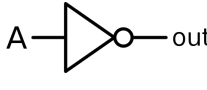
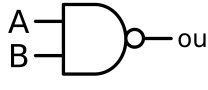
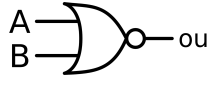
Con due variabili x_0 e x_1 le funzioni fondamentali che bisogna conoscere sono le seguenti:

x_0	x_1	AND	OR	XOR	NAND	NOR
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	1	0
1	1	1	1	0	0	0

AND (x_0, x_1) = $x_0 \cdot x_1$ Prodotto logico

OR (x_0, x_1) = $x_0 + x_1$ Somma logica (Nota differisce da quella binaria)

Le funzioni fondamentali hanno anche una rappresentazione grafica

AND	OR	NOT	NAND	NOR
				

Le funzioni OR e AND sono **associative** per questo posso effettuare somme di prodotti e prodotti di somme contemporaneamente (nella stessa porta).

La difficoltà sta nello scrivere sotto forma di espressione una funzione binaria di variabile binaria, a volte per ottenere l'espressione è semplice usare le varie porte, ma a volta può risultare complesso.

La funzione **AND** ha come valore 1 se solo se tutti gli operandi sono uguali ad 1 , vale 0 negli altri casi.

La funzione **OR** vale 0 se solo se tutti gli operandi sono uguali a 0, 1 negli altri casi.

Una particolare funzione è lo XOR (detto anche OR esclusivo) , tale funzione vale 1 quando c'è almeno un termine diverso dagli altri e 0 quando i termini sono tutti uguali.

x_0	x_1	XOR
0	0	0
0	1	1
1	0	1
1	1	0

2.2 MINTERMINI E MAXTERMINI

I mintermini sono prodotto in cui tutte le variabili compaiono o in forma vera o in forma complementata. Le funzione mintermine valgono 1 in un solo punto e 0 negli altri.

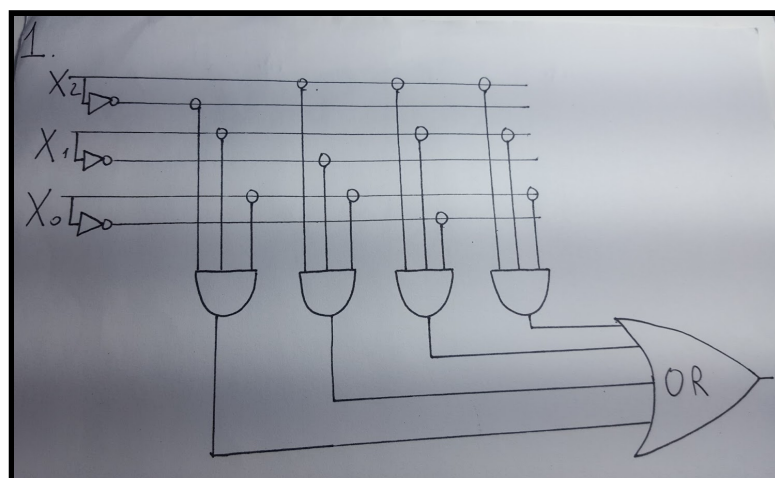
x_2	x_1	x_0	f		m3	m5	m6	m7
0	0	0	0		0	0	0	0
0	0	1	0		0	0	0	0
0	1	0	0		0	0	0	0
0	1	1	1		1	0	0	0
1	0	0	0		0	0	0	0
1	0	1	1		0	1	0	0
1	1	0	1		0	0	1	0
1	1	1	1		0	0	0	1

Scomposizione , $f = m3 + m5 + m6 + m7$.

$$m3 = \bar{x}_2 x_1 x_0 \quad m5 = x_2 \bar{x}_1 x_0 \quad m6 = x_2 x_1 \bar{x}_0 \quad m7 = x_2 x_1 x_0$$

La funzione f è una somma di prodotti (SOP Sum of products).

Rappresentazione di grafica della funzione di f :



I maxtermini sono somme in cui tutte le variabili compaiono o in forma vera o in forma negata. Le funzioni maxtermini valgono 0 in un solo punto e uno in tutti gli altri.

x_2	x_1	x_0	f		$m1$	$m2$	$m6$
0	0	0	1		1	1	1
0	0	1	0		0	1	1
0	1	0	0		1	0	1
0	1	1	1		1	1	1
1	0	0	1		1	1	1
1	0	1	1		1	1	1
1	1	0	0		1	1	0
1	1	1	1		1	1	1

Scomposizione , $f = m1 \times m2 \times m6$.

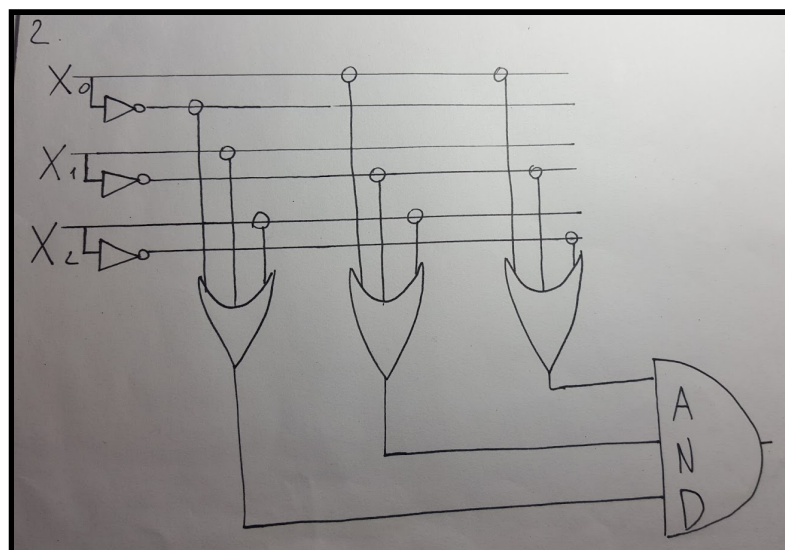
$$m1 = x_2 x_1 \bar{x}_0$$

$$m2 = x_2 \bar{x}_1 x_0$$

$$m6 = \bar{x}_2 \bar{x}_1 x_0$$

La funzione f è un prodotto di somme (POS Products of sum).

Rappresentazione di grafica della funzione di f :



Definizione di Espressione :

- 1) Variabili e costanti sono espressioni
- 2) Se E_1 e E_2 sono espressioni lo sono anche $E_1 \text{ AND } E_2$, $E_1 \text{ OR } E_2$, $\overline{E_1}$, $\overline{E_2}$

Definizione di rete logica :

- 1) I terminali input sono reti logiche ($x_2 \ x_1 \dots$)
- 2) Se N_1 e N_2 sono reti logiche lo sono anche $N_1 \text{ AND } N_2$, $N_1 \text{ OR } E_2$, $\overline{N_1}$, $\overline{N_2}$

NOTA IMPORTANTE

Le reti logiche non hanno feedback, non è quindi possibili retroazione con loro stesse, la retroazione si verifica quando uno dei due input necessari al calcolo della funzione è prodotto da se stessa, vedi flip flop (4.3) . La retroazione è possibile nei dispositivi di memoria.

NOTA IMPORTANTE

Qualsiasi funzione binaria di variabile binaria è combinazione delle funzioni AND , OR e NOT, il quale è detto **insieme funzionalmente completo**.

2.3 ALGEBRA DI COMMUTAZIONE

- 1) $\overline{0} = 1 \quad \overline{1} = 0$
- 2) $X \cdot 1 = X \quad X \cdot 0 = 0$
- 3) $X + 1 = 1 \quad X + 0 = X$
- 4) $\overline{\overline{X}} = X$
- 5) $X \cdot Y = Y \cdot X$
- 6) $X + Y = Y + X$
- 7) $X \cdot X = X^2 = X$
- 8) $X \cdot (Y + Z) = X \cdot Y + X \cdot Z$
- 9) $X + Y \cdot Z = (X + Y) \cdot (X + Z)$

La proprietà 9) vale solo nell'algebra di commutazione e non in quella usuale.

2.4 LEGGI DI DE MORGAN

$$1) \quad \overline{x \cdot y} = \bar{x} + \bar{y}$$

$$2) \quad \overline{x + y} = \bar{x} \cdot \bar{y}$$

$$3) \quad \overline{\overline{x \cdot y}} = \overline{\bar{x} + \bar{y}}$$

$$x \cdot y = \overline{\bar{x} + \bar{y}} \Rightarrow \text{AND} = \text{OR e NOT combinati}$$

$$4) \quad \overline{\overline{x + y}} = \overline{\bar{x} \cdot \bar{y}}$$

$$x + y = \overline{\bar{x} \cdot \bar{y}} \Rightarrow \text{OR} = \text{AND e NOT combinati}$$

La sola porta NAND comprende tutte le funzioni binaria di variabile binaria, infatti da sola costituisce un insieme funzionalmente completo, discorso analogo vale per la NOR (vedi proprietà 3) e 4)).

3. COMPONENTI FONDAMENTALI DI BASE

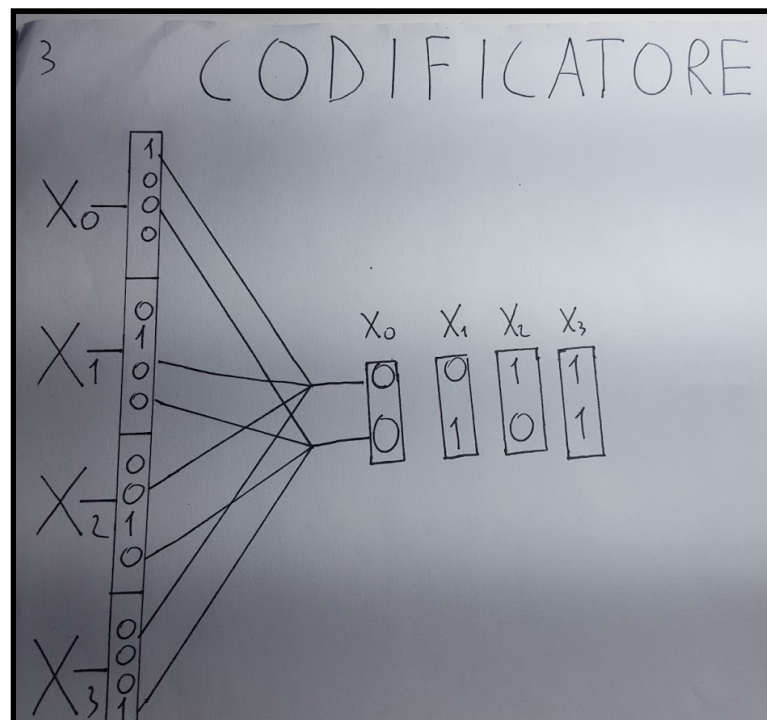
3.1 ENCODER (CODIFICATORE)

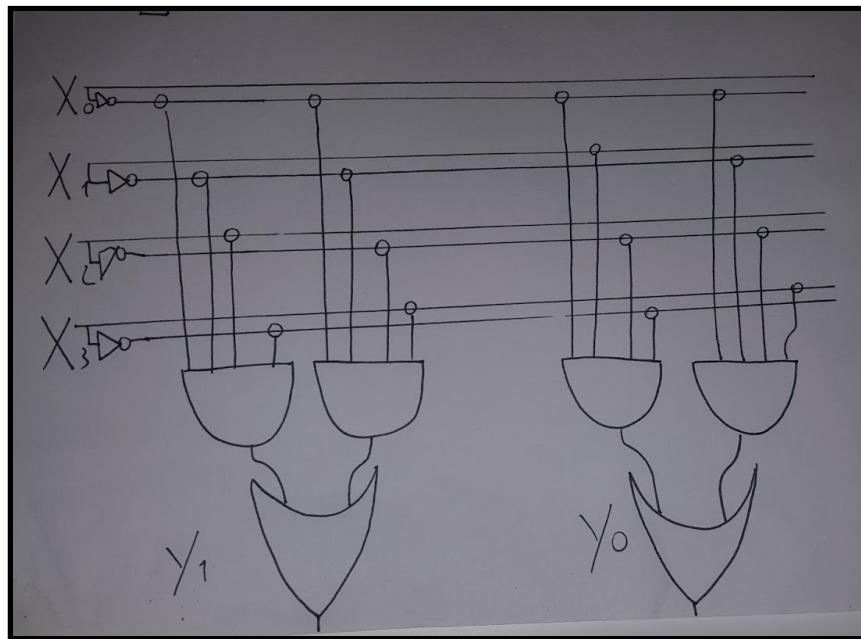
Il codificatore è un dispositivo (nello specifico un circuito combinatorio) che prende in input una sequenza di n bit in cui un solo bit può essere uno e restituisce un valore di $\log(n)$ bit ; dà una rappresentazione binaria dell'indice in cui viene mandato 1.

Esempio di codificatore 4 a 2 :

Tavola di verità

x_3	x_2	x_1	x_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1





3.2 DECODER (DECODIFICATORE)

Il decodificatore è un dispositivo (nello specifico un circuito combinatorio) che prende in input una sequenza di n bit e restituisce un valore di 2^n bit; dà una sequenza binaria dove mette tutti bit a 0 e un bit a 1 ovvero quello di indice corrispondente al numero rappresentato nella sequenza data in input.

Esempio di decodificatore a 2 a 4 :

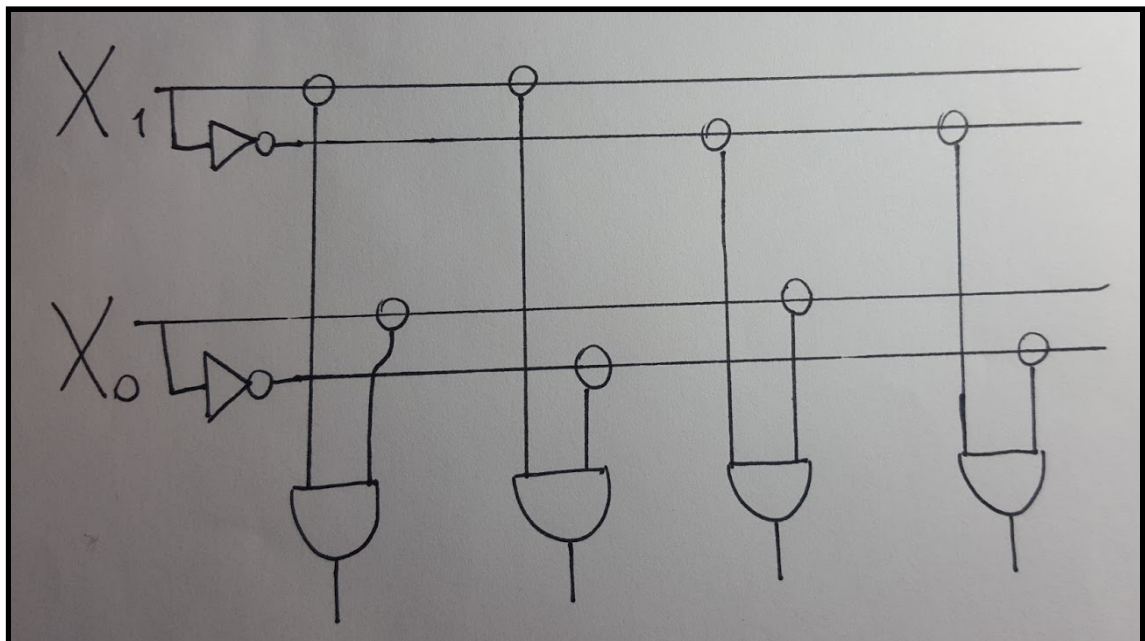
Tavola di verità

x_1	x_0	y_3	y_2	y_1	y_0
0	0	0	0	0	1
0	1	0	0	1	1
1	0	0	1	0	0
1	1	1	0	0	0

4.

DECODER _{2 to 4}

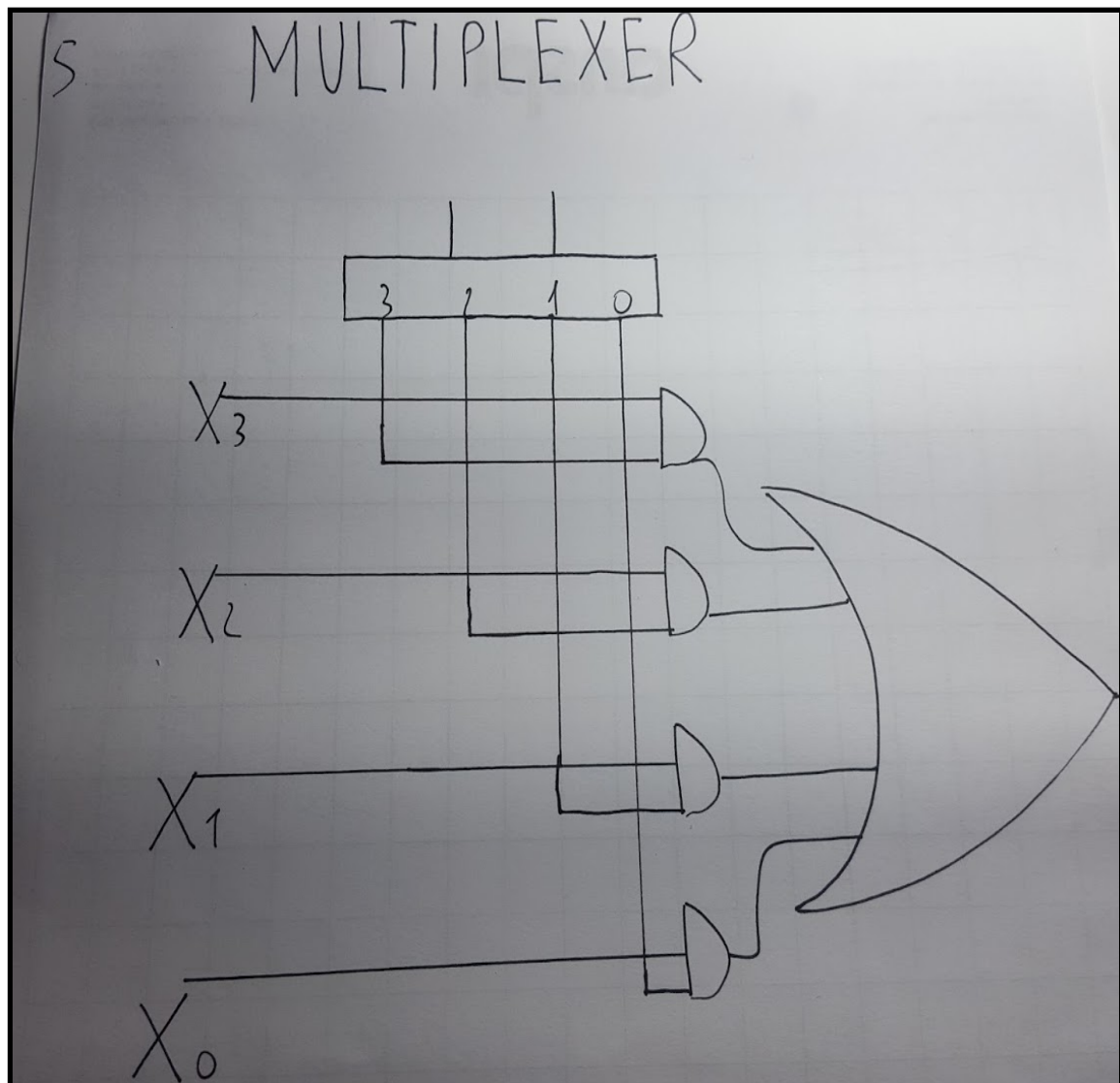
X_1	X_0	
0	0	\longrightarrow 0 0 0 1
0	1	\longrightarrow 0 0 1 0
1	0	\longrightarrow 0 1 0 0
1	1	\longrightarrow 1 0 0 0



3.3 MULTIPLEXER

Il multiplexer è un dispositivo (nello specifico un circuito combinatorio) che prende n bit di controllo e 2^n bit di input e darà in output 1 solo bit; il bit in output sarà il bit della sequenza di bit di indice uguale al numero rappresentato nei bit di controllo.

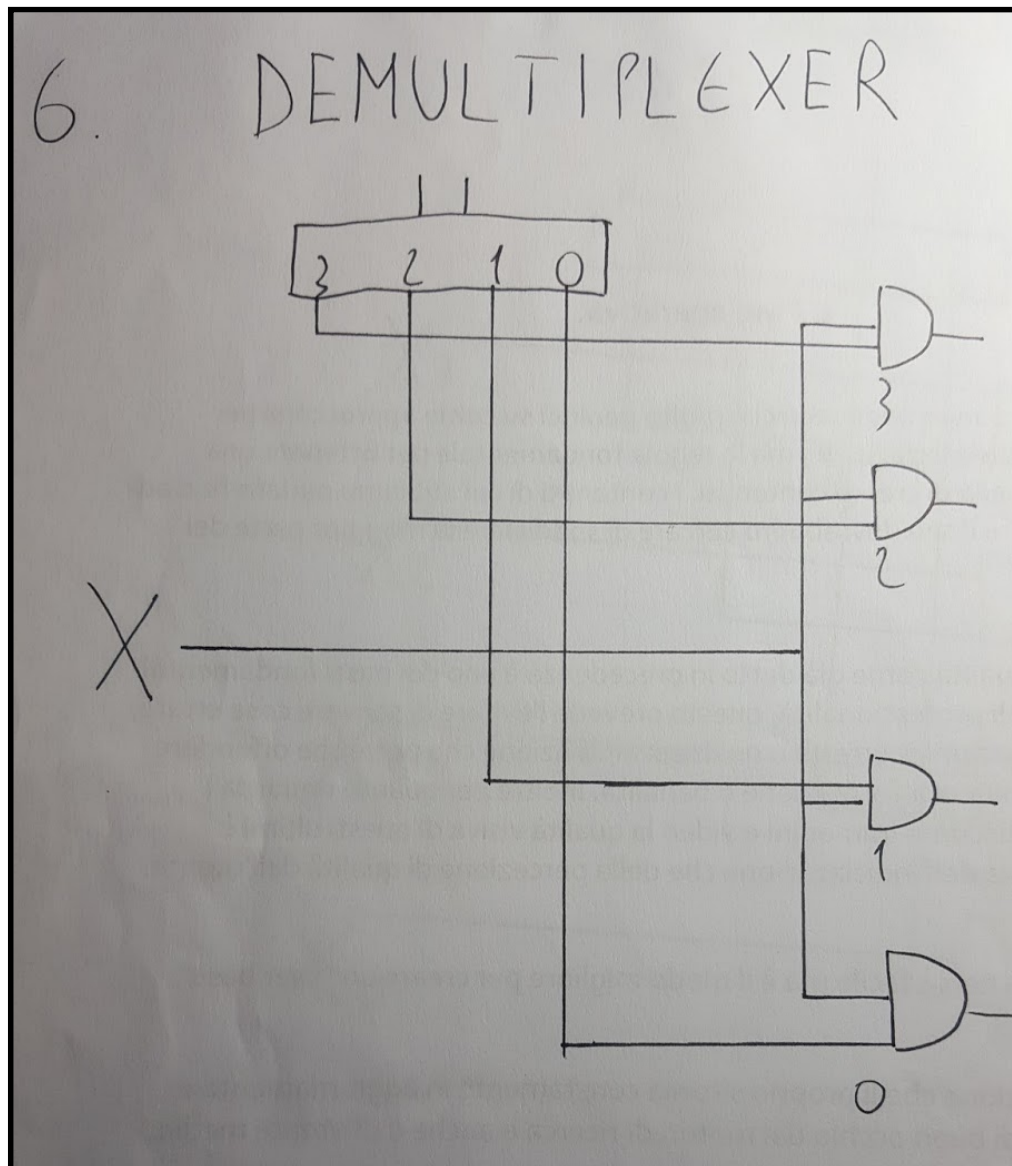
Esempio di multiplexer a 4 a 1 (2 bit di controllo) :



3.4 DEMULTIPLEXER

Il demultiplexer è un dispositivo (nello specifico un circuito combinatorio) che prende n bit di controllo e 1 bit di input e darà in output 2^n bit; i bit in output saranno tutti quanti a 0 tranne il bit di indice uguale al numero rappresentato nei bit di controllo, il quale avrà lo stesso valore del bit dato in input.

Esempio di demultiplexer a 1 a 4 (2 bit di controllo) :



3.5 SOMMATORE

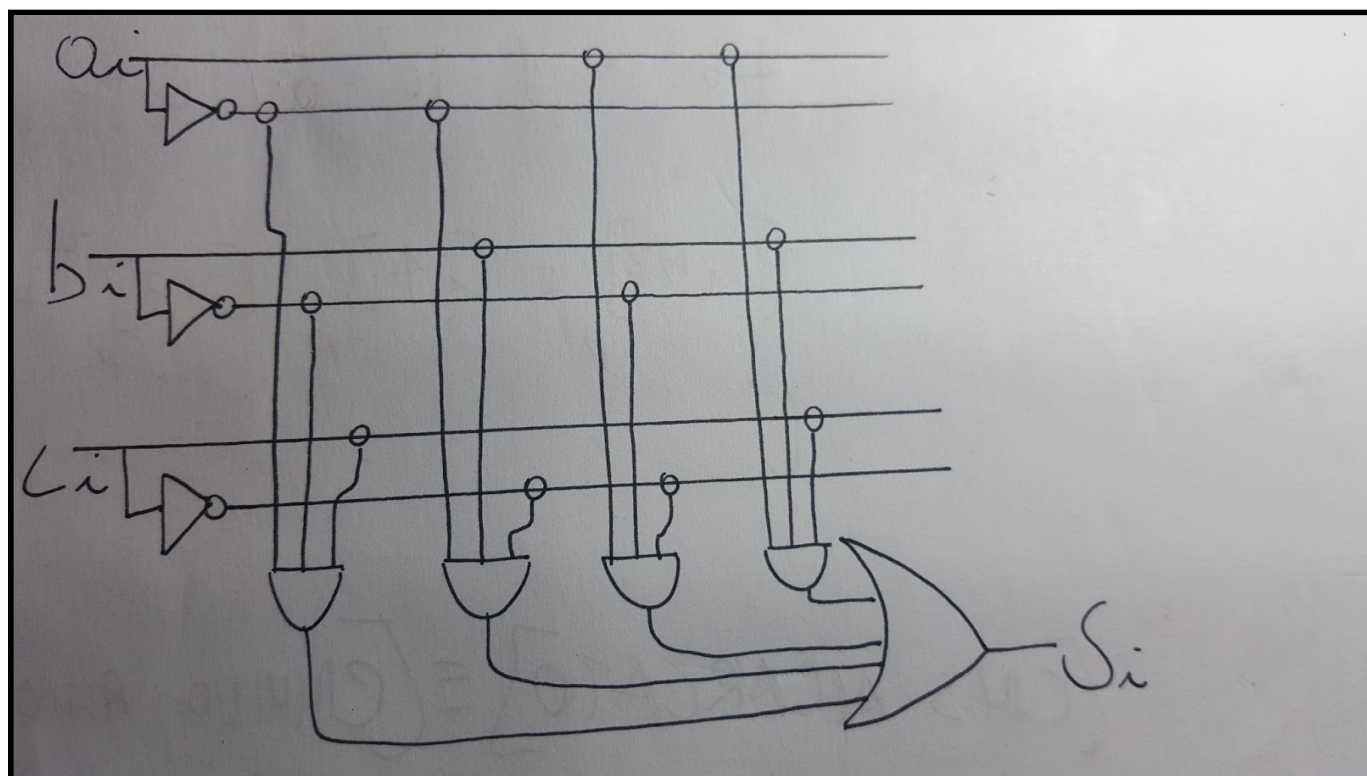
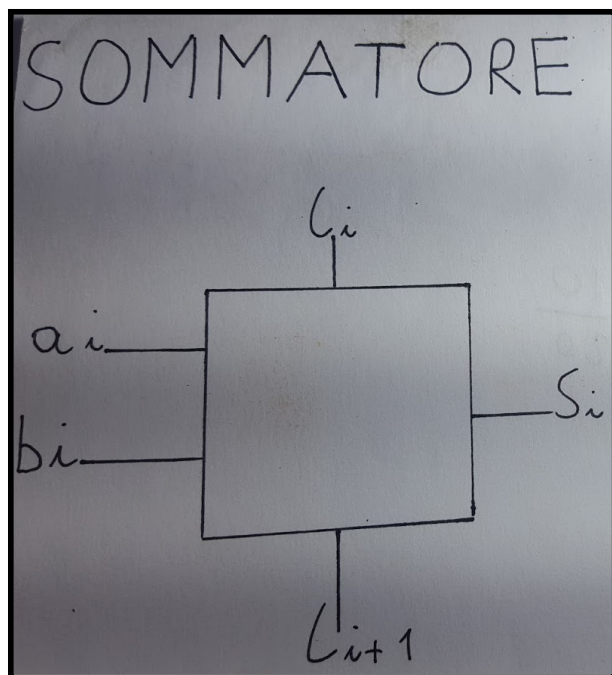
Il sommatore è un dispositivo (nello specifico un circuito combinatorio) che prende in input 3 bit , due operandi e il resto prodotto dalla somma dei bit di posizione precedente; dà in output due bit uno che rappresenta la somma tra i due operandi e il secondo rappresenta il resto prodotto dalla somma.

Tavola di verità

a_i	b_i	c_i	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Scomposizione in mintermini di S				Scomposizione in mintermini di C			
m1	m2	m4	m7	m3	m5	m6	m7
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	1

Rappresentazione grafica del sommatore



4. PROGETTAZIONE MIPS CICLO SINGOLO

4.1 ISTRUZIONI E FORMATI

Quando si progetta un processore implicitamente si decide il set di istruzioni che è capace di eseguire. Un primo tentativo di traduzione del linguaggio macchina intrinseco al processore è il linguaggio assembler. La sintassi di questo linguaggio ha delle regole ben precise che variano in base alla tipologia di istruzione.

Ogni istruzione assembly per essere eseguita dal processore viene tradotta dalla equivalente istruzione in linguaggio macchina rappresentata su 32 bit. I 32 bit dell'istruzione hanno una suddivisione ben precisa che varia in base al formato dell'istruzione da tradurre.

INDIRIZZI

Per indicare l'indirizzo di uno dei 32 registri del banco dei registri necessitiamo di 5 bit in binario puro.

La memoria è costituita da 2^{32} locazioni da 1 byte l'una, noi però necessitiamo di 4 byte per costituire una "word", per questo si è deciso di dividere la memoria in 4 blocchi da 2^{30} locazioni da 1 byte l'una e di assegnare un indirizzo ad ogni parola che varia di modulo 4.

L' assembler MIPS presenta nella versione che studieremo 3 formati :

FORMATO R

OP	RS	RT	RD	VOID	FUNCTION
6 BIT (31 - 26)	5 BIT (25 - 21)	5 BIT (20 - 16)	5 BIT (15 - 11)	5 BIT (6 - 10)	6 BIT (5 - 0)

- **OP**

I primi 6 bit delle istruzioni sono riservati al codice operativo che permette al processore di capire di quale istruzione si tratta e di agire di conseguenza.

- **RS**

Il campo individuato da RS specifica quale dei 32 registri del banco dei registri è quello sorgente.

- **RT**

Il campo individuato da RT specifica quale dei 32 registri del banco dei registri è quello target.

- **RD**

Il campo individuato da RD specifica quale dei 32 registri del banco dei registri è quello destinazione.

- **VOID**

Il campo void come il nome lascia presagire che è un campo vuoto.

- **FUNCTION**

Il campo function contiene i bit necessari ad indicare al processore (ALU) quale operazione eseguire.

FORMATO I

OP	RS	RT	COSTANTE
6 BIT (31 - 26)	5 BIT (25 - 21)	5 BIT (20 - 16)	16 BIT (15 - 0)

- **OP**

I primi 6 bit delle istruzioni sono riservati al codice operativo che permette al processore di capire di quale istruzione si tratta e di agire di conseguenza.

- **RS**

Il campo individuato da RS specifica quale dei 32 registri del banco dei registri è quello sorgente.

- **RT**

Il campo individuato da RT specifica quale dei 32 registri del banco dei registri è quello target.

- **COSTANTE**

Il campo costante contiene un numero a 16 bit in

FORMATO J

OP	COSTANTE
6 BIT (31 - 26)	26 BIT (25 - 0)

- **OP**

I primi 6 bit delle istruzioni sono riservati al codice operativo che permette al processore di capire di quale istruzione si tratta e di agire di conseguenza.

- **COSTANTE**

Il campo costante indica l'indirizzo dell'istruzione a cui saltare.

ISTRUZIONI ARITMETICHE ADD, SUB, SLT, AND, OR

Le istruzioni aritmetiche sono di tipo R e servono per fare operazioni con i bit. I contenuti individuati dai campi RS e RT sono gli operandi che verranno mandati all'ALU, il registro individuato dal campo RD è invece il registro nel quale il processore andrà a scrivere il risultato, il campo FUNCTION invece conterrà i bit che indicheranno all'ALU quale operazione eseguire.

ADD L'istruzione ADD effettua la somma in complemento a due dei due operandi indicati.

SUB L'istruzione SUB effettua la somma in complemento a due del primo operando indicato con l'opposto del secondo.

SLT L'istruzione SLT effettua un confronto e vale 1 se il primo operando è minore del secondo e 0 negli altri casi. Poiché vogliamo vedere se $A < B$ ciò equivale a vedere se A

- $B < 0$ effettua una differenza tra il primo e il secondo e prende dalla cella 31 il risultato del sommatore, che equivale al bit di segno della sottrazione, questo bit sarà il risultato della SLT.

AND L'istruzione AND effettua l'and bit a bit dei due operandi inviati.

OR L'istruzione OR effettua l'or bit a bit dei due operandi inviati.

ISTRUZIONI ARITMETICHE CON COSTANTE

Le istruzioni aritmetiche con costante sono di tipo I e servono per fare operazioni in modo analogo alle istruzioni aritmetiche di tipo R, la differenza sta negli operandi poiché gli operandi mandati all'ALU saranno il contenuto del registro individuato dal campo RS e il numero a 16 bit in complemento a 2 espresso nel campo costante, il registro individuato dal campo RT sarà il registro nel quale il processore andrà a scrivere il risultato. **NOTA** non essendo presente il campo function l'alu capirà quale operazione eseguire in base al campo OP e il processore capirà e comunicherà internamente all'ALU quale delle operazioni eseguire.

ISTRUZIONI LW, SW

L'istruzione LW e SW sono di tipo I e servono rispettivamente per caricare e scaricare dati dalla memoria dati.

L'istruzione LW serve per leggere un dato dalla memoria dati e scriverlo nel banco registri. Poiché gli indirizzi della memoria sono tutti multipli di 4 gli ultimi due bit degli indirizzi sono entrambi 0. Per individuare una particolare cella della memoria l'LW somma il registro individuato dal campo RS alla costante a 16 bit estesa di segno a 32, il campo RT invece individua il registro dove andare a scrivere il valore contenuto nella cella individuata.

L'istruzione SW calcola l'indirizzo in modo analogo alla LW, il campo RT invece individua il dato da scrivere nella cella individuata.

ISTRUZIONE BEQ BNE

L'istruzione BEQ effettua una verifica tra i contenuti dei registri individuati dai campi RS e RT, effettua la differenza tra i due operandi e prende il bit zero generato dall'ALU (ovvero la NOR di tutti i bit di risultato), e la manda in AND con un bit di controllo (il bit , ovvero il primo bit di branch), quando il bit zero vale uno vuol dire che bisogna effettuare il salto. I 16 bit della costante vengono estesi a 32, moltiplicati per 4 e sommati al valore del program counter + 4.

L'istruzione BNE funziona in modo analogo alla BEQ con la differenza che salta solo se i due operandi sono diversi. (quindi manda in AND il negato del bit zero e un bit di

controllo , ovvero il secondo bit di branch).

ISTRUZIONE JUMP E JAL

L'istruzione JUMP è di tipo J e costituisce il salto incondizionato. Poiché le 2^{32} locazioni della memoria istruzioni sono divise in 16 blocchi di 2^{28} locazioni individuate dai primi 4 bit dell'istruzione scalo i 26 bit della costante di due posti così da averne 28 e gli affianchiamo i 4 bit dell'istruzione affinché l'indirizzo di salto sia relativo al blocco corretto. L'istruzione JAL a differenza dell'istruzione DA CONTINUARE

ISTRUZIONE JR

L'istruzione JR è di formato R ma utilizza solo 11 bit ovvero il campo del codice operativo e il campo RS, funziona in modo analogo all'istruzione J solo che salterà all'indirizzo scritto nella costante

4.2 UNITA LOGICO ARITMETICA (ALU)

L'ALU è l'unica unità del processore capace di manipolare bit; tale unità prende in input due operandi A e B a 32 bit (in complemento a 2) e 4 bit di controllo che definiscono l'operazione che l'alu deve effettuare.

L'ALU internamente è composta da 32 celle (da 0 a 31) ognuna delle quali agisce sui bit di indice pari all'indice della cella, ogni cella dell'ALU effettua sempre 4 operazioni , AND, OR, SOMMA e SLT, e contiene internamente un multiplexer che prende in input i risultati delle 4 operazioni e in base a due bit **op1** e **op2** di controllo mandati all'ALU decide quale risultato mandare. (nota : tutte le celle dell'ALU condividono i bit di controllo), ogni cella quindi produrrà il bit di risultato e il bit di riporto.

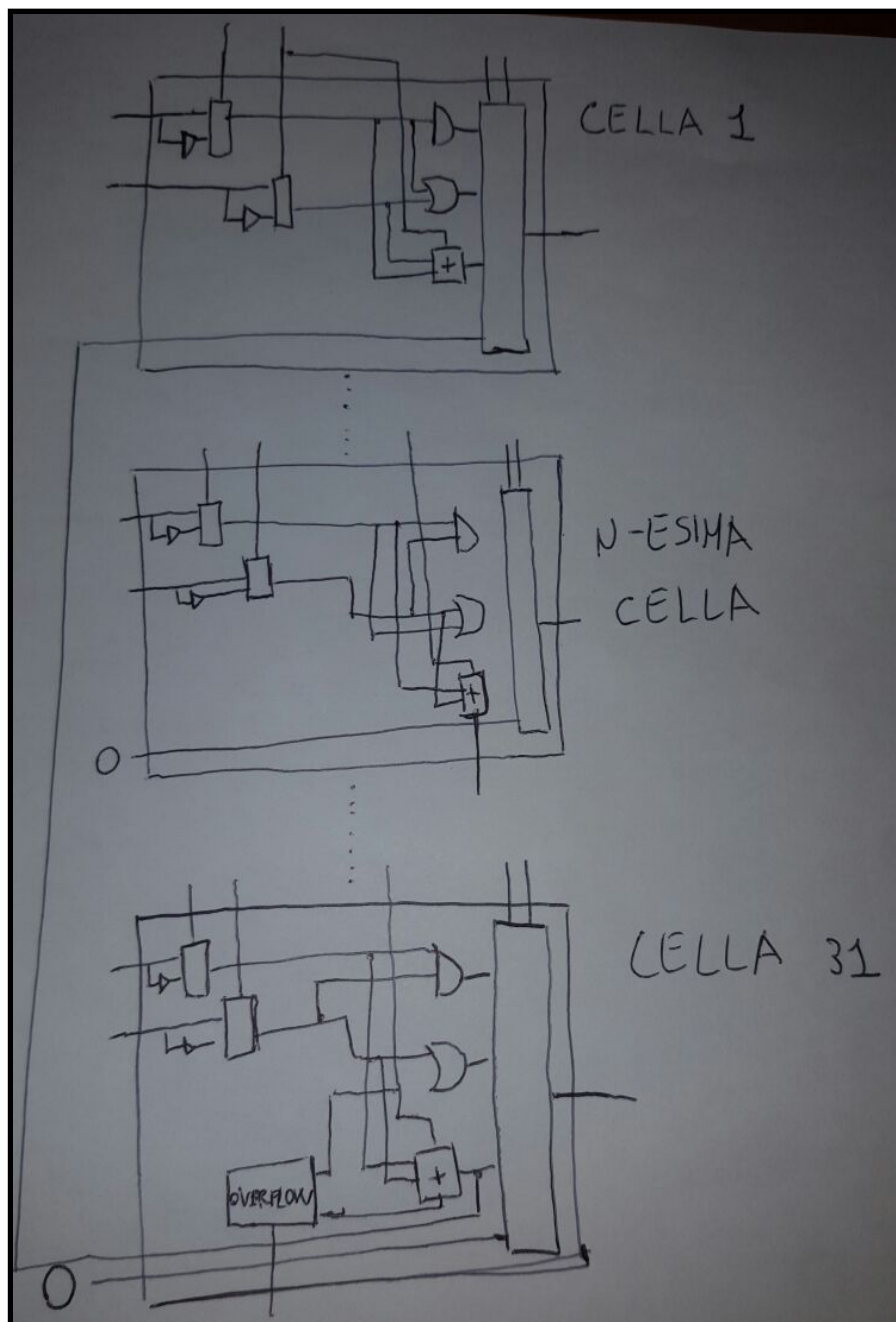
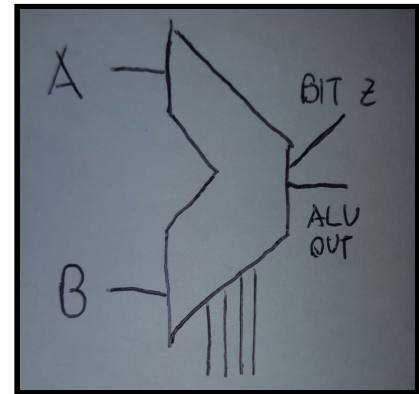
Come detto in precedenza due dei 4 bit di controllo sono utilizzati per decidere quale risultato mandar fuori dall'ALU, gli altri due sono **negaA** e **negaB** e servono a far arrivare rispettivamente il primo e il secondo operando in forma negata, infatti tali bit saranno bit di controllo di un multiplexer che deciderà se mandar fuori il bit di A in forma vera o in forma negata, discorso analogo per B.

La prima cella dell'ALU avrà il bit di riporto precedente collegato al bit di controllo negaB, questo per dare la possibilità di sommare uno così permettendo il complemento di B.

L'ultima cella dell'ALU ha due particolarità, la prima riguarda l'**SLT** poiché tale cella manderà indietro il bit prodotto dal sommatore e lo collegherà al bit di less della cella 0, tutte le altre celle avranno il bit di less settato a 0; la seconda particolarità è legata all'**overflow** la cella 31 si occuperà di prendere il riporto della cella 30 e della cella

31 e di mandarli in input ad una particolare unità che si occuperà dell'overflow (per controllare se l'overflow si sia verificato o meno abbiamo bisogno di una porta xor).

Segue la rappresentazione grafica dell'ALU :



4.3 FLIP FLOP

Il flip flop è il primo dispositivo in cui compare **retroazione**, cioè utilizza come input uno dei propri output prodotti nello stato precedente.

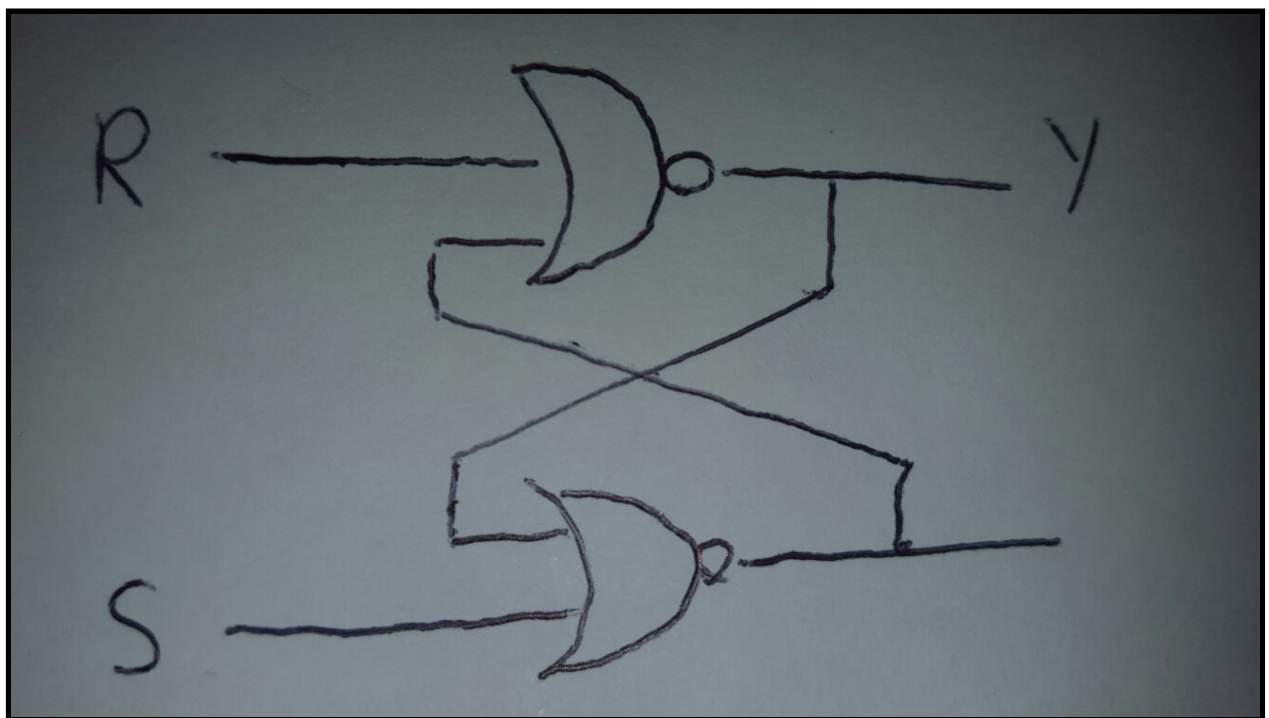
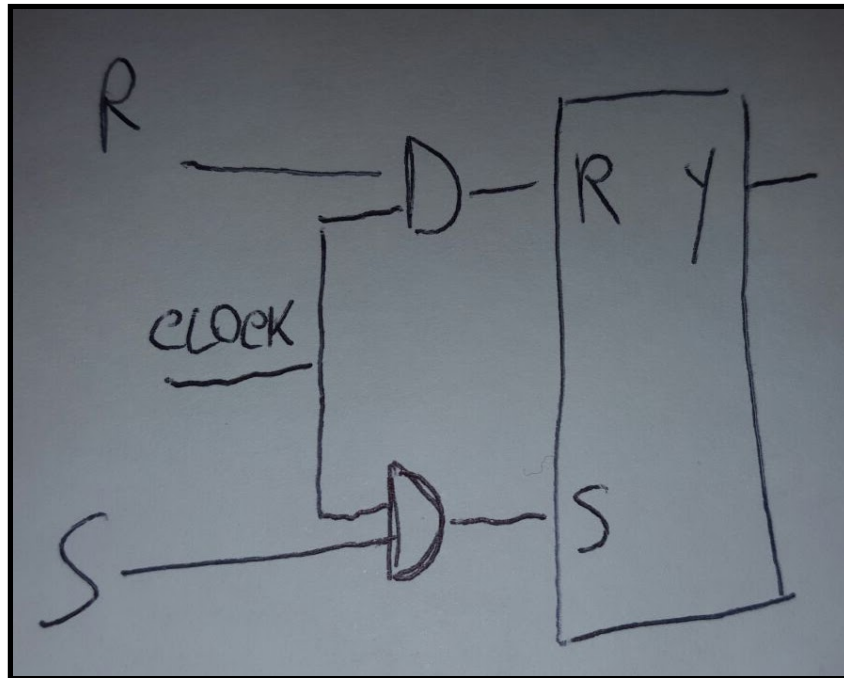
Abbiamo due tipologie di flip-flop:

1) Flip-flop SR

Il flip-flop di tipo SR prende 3 input, il principale è quello di clock che viene messo ad 1 solo quando vogliamo modificare lo stato del flip-flop. Il bit di set viene mandato ad 1 quando vogliamo mettere il flip flop ad 1, quello di reset viene mandato ad 1 quando vogliamo mettere il flip flop a 0, i bit non possono avere valore 1 nello stesso momento, in quel caso il flip-flop avrebbe un comportamento non definito; precisiamo che ha senso mandare bit di set e reset per variare lo stato solo quando il bit di clock è ad 1 poiché quest'ultimo viene messo in AND con i due precedenti.

Tavola caratteristica

R	S	Y^{T-1}		Y^T
Input di stabilità				
0	0	0		0
0	0	1		1
Input di set				
0	1	0		1
0	1	1		1
Input di reset				
1	0	0		0
1	0	1		0



2) Flip-flop D

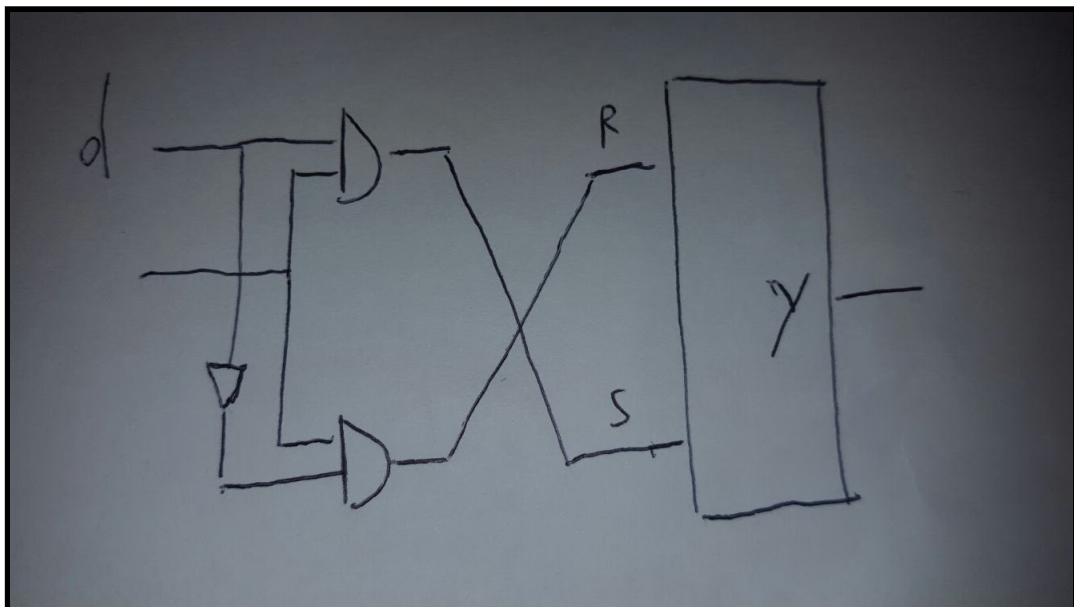
Come visto precedentemente i bit di Set e Reset non possono avere valore 1 contemporaneamente e per questo gli input diventano 2 dai 3 originari del tipo RS.

Il bit di clock svolge lo stesso ruolo svolto precedentemente e viene mandato in AND con il bit D e con il suo negato che equivalgono ai bit di Set e di Reset.

La variazione di stato del flip flop dopo l'abilitazione del Clock non è immediata e per questo questi flip-flop vengono identificati con il tipo D (Delay), proprio perché c'è un leggero ritardo prima che ci sia un effettivo cambio di stato.

Tavola caratteristica

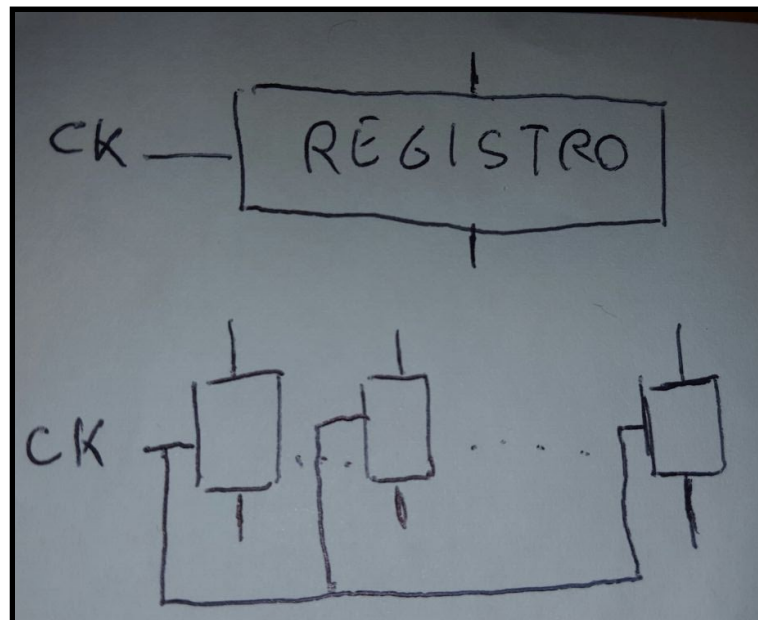
D	Y ^{T-1}		Y ^T
Input di reset			
0	0		0
0	1		0
Input di set			
1	0		1
1	1		1



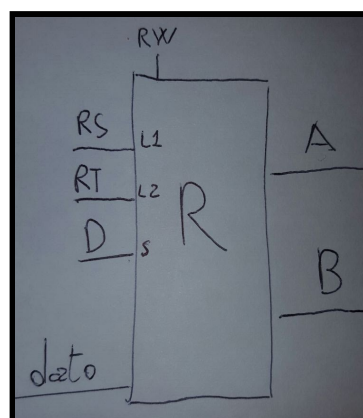
4.4 BANCO DEI REGISTRI

Il banco dei registri è un dispositivo di memoria collegato direttamente all'ALU, prende in input 3 valori a 5 bit e dà in output due valori a 32 bit. Tale dispositivo prende in input un bit di controllo **reg write** che servirà a disabilitare (0) / abilitare (1) la scrittura; due dei valori presi in input sono RS e RT che rappresentano l'indirizzo dei registri cui interno vi sono i due numeri a 32 bit che devono essere prodotti in output. Il terzo valore è RD e cioè il registro destinazione dove nel caso di una istruzione aritmetica di tipo R o di tipo I tale registro è il registro all'interno del quale verrà memorizzato il risultato prodotto dall'ALU, nell'istruzione LW invece non essendo presente il campo RD, l'indirizzo del registro destinazione è quello del campo RT.

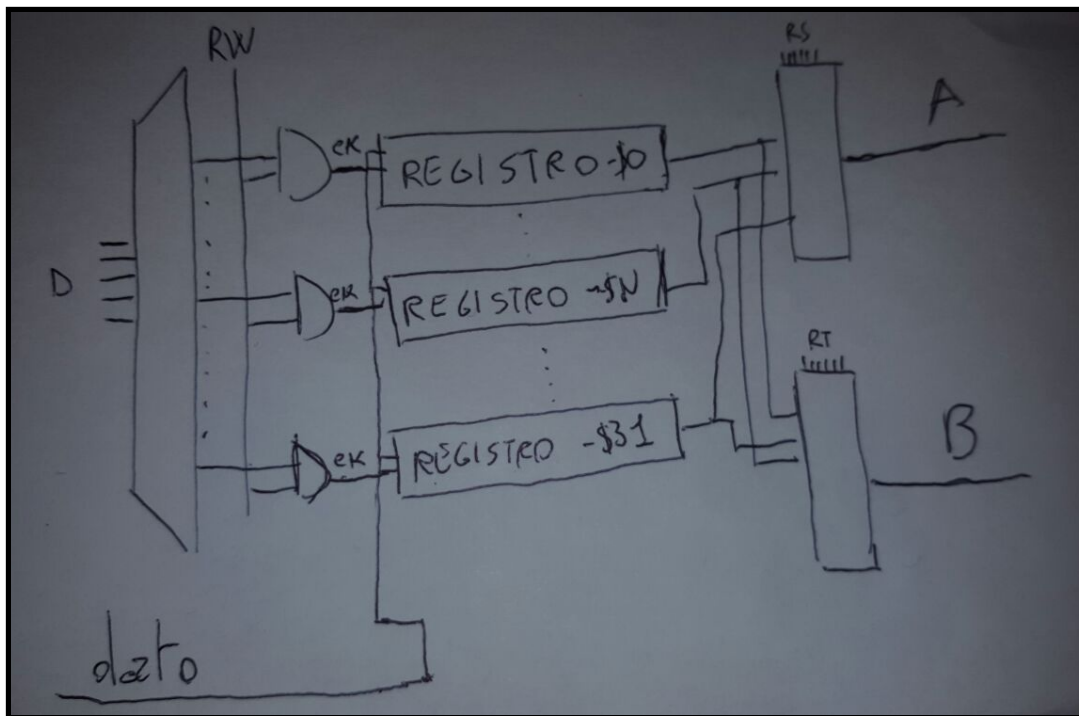
Ogni registro è composto da 32 flipflop di tipo D che condividono il bit di clock :



Una prima vista ad alto livello del banco dei registri quindi è la seguente :



Nello specifico il progetto del banco dei registri prevede internamente i seguenti componenti :



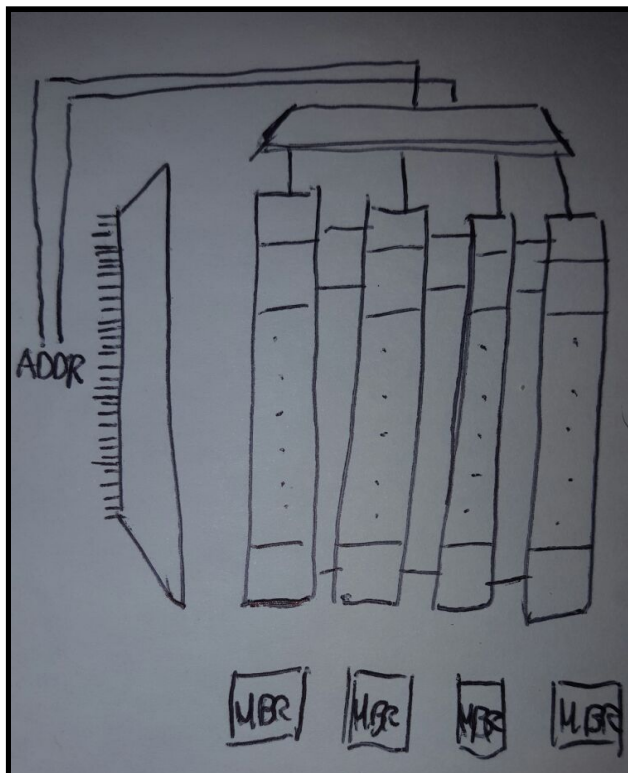
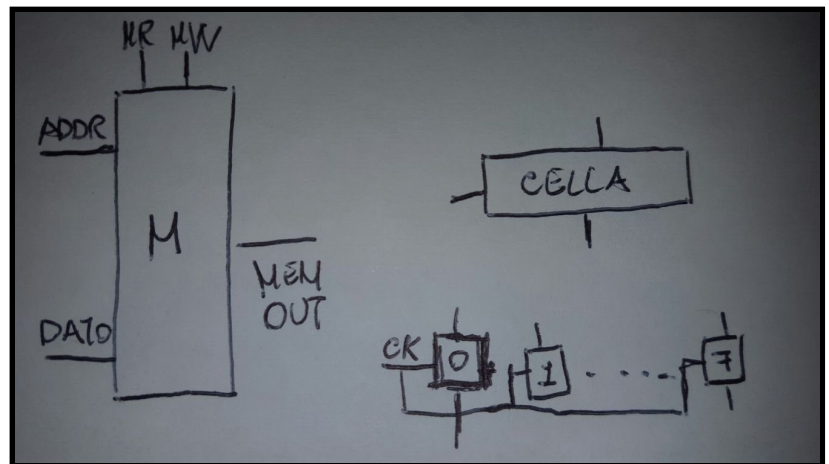
IL dato a 32 bit che si vuole scrivere viene mandato a tutti i registri, il bit di clock dei vari registri viene determinato dal risultato del decoder e dal bit rw messi in AND.

I multiplexer per la lettura in realtà sono 32 multiplexer e ognuno prende in input l'i-esimo bit del registro. I bit di controllo dei multiplexer sono 5 e sono proprio i valori a 5 bit di RS e RT.

4.5 MEMORIA ISTRUZIONI / MEMORIA DATI

La memoria è un altro dei dispositivi di memorizzazione del processore. La memoria contiene 2^{32} locazioni, ogni locazione è composta da 1 byte e quindi da 8 flipflop in sequenza che condividono il bit di clock. Gli indirizzi di memoria sono a 32 bit per poter individuare potenzialmente qualsiasi locazione della memoria. Si ha però la necessità di individuare una parola, la quale è composta da 32 bit, per questo motivo si è deciso di dividere la memoria in 4 banchi da 2^{30} locazioni, per cui gli indirizzi come vedremo saranno tutti multipli di 4, quindi gli ultimi due bit saranno entrambi a 0.

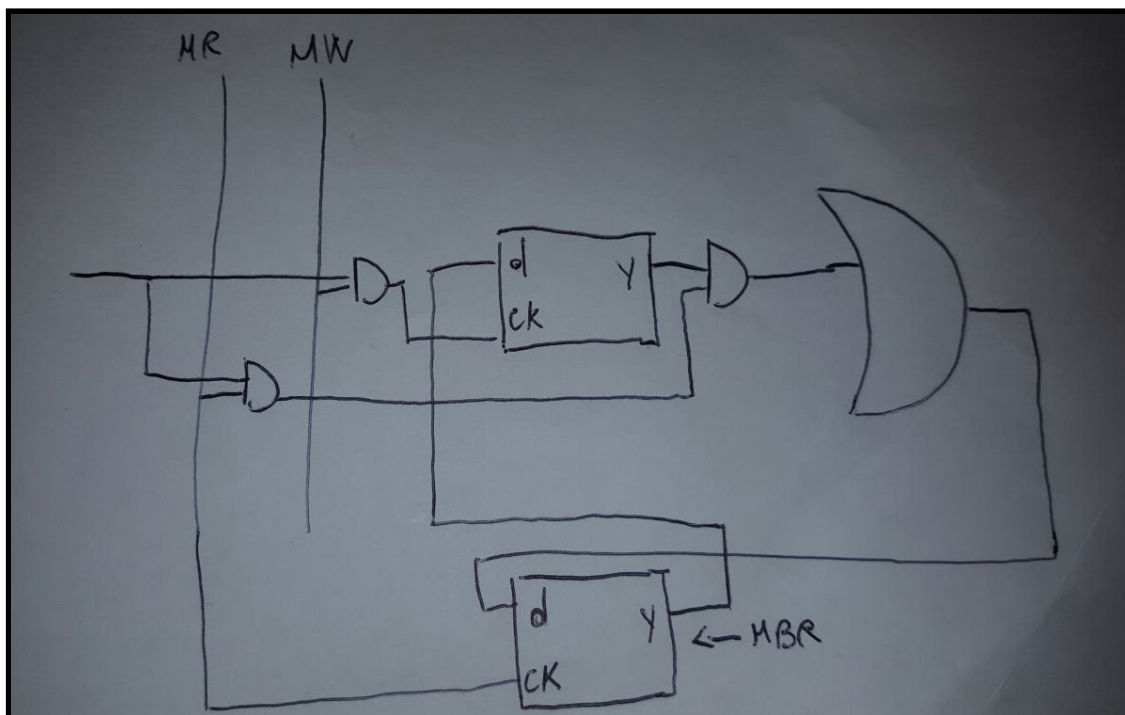
Una prima vista ad alto livello della memoria dati può essere la seguente :



Poiché necessito solo di 30 bit dei 32 dell'indirizzo, gli ultimi 2 bit quelli cioè meno significativi vengono mandati in input ad un ulteriore decoder che utilizza questi bit per selezionare uno tra 4 blocchi della memoria :

In fase di scrittura nel mar (memory address register) c'è l'indirizzo del registro dove si vuole scrivere, i 30 bit più significativi di tale indirizzo vengono mandati ad un decoder, ogni bit che esce dal decoder sarà messo in and con il bit di mem writer e il risultato sarà il bit di clock della cella relativa a quel bit. Il dato a 32 bit che si intende scrivere è presente nel registro di buffer, tale MBR (memory buffer register), quindi ogni locazione prende in input i bit dell' MBR, ma solo la locazione che avrà il bit di clock ad 1 lo scriverà.

In fase di lettura nel mar c'è l'indirizzo del registro che si vuole leggere, i 30 bit più significativi di tale indirizzo vengono mandati ad un decoder, ogni bit che esce dal decoder sarà messo in and con l'uscita dell'iesimo flipflop del registro relativo a quel bit, l'uscita di ogni porta and andrà in input ad un grande porta or che prenderà i bit di tutti i flipflop della medesima posizione, è evidente che le porte or sono 32 e l'uscita di queste porte sarà input dell' MBR. Il bit di clock dell'mbr è il bit di mem read. Il dato che si voleva leggere infine quindi è presente nel registro di buffer.



4.6 CONTROL UNIT

Introduciamo adesso un dispositivo che sarà di fondamentale importanza sia nel progetto a ciclo singolo che in quello con registri di pipe: La “Control Unit”.

La Control Unit è un dispositivo combinatorio e ha il ruolo di produrre una determinata quantità di bit che abilitano/disabilitano determinati dispositivi per le fasi successive. La CU prende i 6 bit più significativi dalla memoria istruzioni, i quali corrispondono al codice operativo di una determinata operazione, e produce i bit di controllo corrispondenti affinché ci sia un corretto funzionamento del processore.

Nel ciclo singolo la CU produce esattamente 10 bit di controllo:

01. Reg Write, questo bit abilita la scrittura nel banco dei registri. Validato per istruzioni di tipo “R” e per la “LW” poiché vi è fase di Write-Back.
02. Register Destination, questo bit controlla l’omonimo multiplexer e sceglie se mandare come registro destinazione i bit 25-21 o i bit 20-16 a seconda del formato dell’istruzione (R o I).
03. Alu Source, questo bit controlla l’omonimo multiplexer e sceglie se mandare, come secondo input all’ALU, il contenuto di uno dei registri oppure i bit 15-0 estesi a 32 che corrispondono alla costante C, ciò varia a seconda delle istruzioni (R, I o LW/SW).
04. ALU-OP.1 ALU-OP.2, questi due bit controllano il circuito combinatorio che manda i bit di controllo all’ALU e possono avere i seguenti valori:
 - 00, questa combinazione fa eseguire una somma incondizionata.
 - 01, questa combinazione fa eseguire una differenza incondizionata.
 - 10, questa combinazione fa valere i 6 bit meno significativi dell’istruzione (5-0) non modificando le direttive dell’ALU.
05. Branch, questo bit vale uno solo se l’istruzione da eseguire è una BEQ e viene mandato in AND con il bit Z, il risultato della porta AND controlla un multiplexer che prende all’ingresso 0 il valore di PC+4 e all’ingresso 1 l’indirizzo di salto calcolato precedentemente; è ovvio che il risultato della porta AND sarà 1 solo quando avranno tale valore sia Z che Branch, ciò evita salti non programmati.
 - Eventualmente esiste un secondo bit Branch valido per la BNE il quale viene mandato in AND con il negato del bit Z.
06. J, questo bit vale 1 solo quando si vuole eseguire un salto incondizionato e controlla un determinato multiplexer.
07. MEM READ - MEM WRITE
 - Questi due bit abilitano le funzioni di lettura/scrittura della Memoria Dati e non possono avere valore 1 contemporaneamente.

08. Mem-to-Reg, questo bit vale uno solo quando si ha funzione di Write-Back e permette la scrittura di un determinato dato nel banco dei Registri.

ORGANIZZAZIONE BIT DI CONTROLLO CICLO SINGOLO

	RD	RW	AS	ALU op1	ALU op2	B	J	MR	MW	MTR
R	1	1	0	1	0	0	0	0	0	0
LW	0	1	1	0	0	0	0	1	0	1
BEQ	0	0	0	0	1	1	0	0	0	0
SW	0	0	1	0	0	0	0	0	1	0
J	0	0	0	0	0	0	1	0	0	0
ADDI	0	1	1	0	0	0	0	0	0	0
SUBI	0	1	1	0	1	0	0	0	0	0

4.7 FASI DEL PROCESSORE

Il MIPS è stato diviso concettualmente in fasi, le fasi sono in tutto 5, e sono: FETCH, DEC, EXE, MEM e WRITE-BACK.

1. FETCH

La fase di fetch comprende il prelievo dal program counter dell'indirizzo dell'istruzione da eseguire, il calcolo di $PC + 4$ e il prelievo dell'istruzione da eseguire presente all'indirizzo prelevato dal program counter.

2. DEC

La fase dec prevede la decomposizione dell'istruzione :

I bit 31-26 vengono mandati alla control unit che si occuperà di produrre i bit di controllo necessari all'esecuzione dell'istruzione

I bit 25-21 che vengono mandati al banco dei registri, e rappresentano RS.

I bit 20-16 che vengono mandati al banco dei registri, e rappresentano RT.

In fase dec è presente anche il multiplexer register destination, che prende in input l'omonimo bit di controllo, e il valore di RT e i bit 15-11.

Il banco dei registri infine produrrà A e B.

Sempre in fase dec viene calcolato il potenziale indirizzo utilizzato dal salto incondizionato e vengono estesi a 32 i bit del campo costante.

3. EXE

In fase EXE l'ALU prenderà in input A e il risultato del multiplexer ALU SOURCE che prende in input l'omonimo bit di controllo, B e il campo costante esteso a 32. L'ALU è controllata da un unità (composta da un circuito combinatorio) che prende in input i 6 bit del campo function e 2 bit di controllo ALU op1 e ALU op2. L'alu produrrà il risultato dell'operazione che gli è stata indicata e il bit di z che servirà per il salto condizionato. Inoltre in fase exe viene anche calcolato il potenziale indirizzo di salto condizionato e vi sono i multiplexer di salto condizionato e incondizionato.

4. MEM

La fase mem è l'unica fase che ha accesso alla memoria dati, la memoria dati è controllata dai bit mem read e mem write, Prende in input il risultato dell'ALU che diventerà l'indirizzo della cella della memoria istruzione dove si vorrà scrivere o leggere in base se l'istruzione è una SW o una LW. Il valore da scrivere nel caso di una SW è l'operando B prodotto dal banco dei registri.

5. WRITE-BACK

La fase write-back permette di scrivere un dato nel banco dei registri, le uniche istruzioni che utilizzano questa fase sono LW e R. La fase write back presenta il multiplexer mem to reg che prende in input l'omonimo bit di controllo, il dato prodotto dall'ALU e il dato prodotto dal banco dei registri. Il risultato di tale multiplexer sarà mandato come dato da scrivere al banco dei registri.

ORGANIZZAZIONE FASI DEL PROCESSORE MIPS CICLO SINGOLO

HA LA FASE	NON HA LA FASE

	FETCH	DEC	EXE	MEM	WRITE BACK
R					
LW					
BEQ					
SW					
J					
ADDI					
SUBI					

4.8 INTRODUZIONE SALTI

Come abbiamo visto tra le istruzioni che il MIPS prevede ci sono 2 tipi di salto: il salto incondizionato (J) e il salto condizionato (BEQ/BNE). L'introduzione di tali salti nel MIPS a ciclo singolo riguarda la fase EXE.

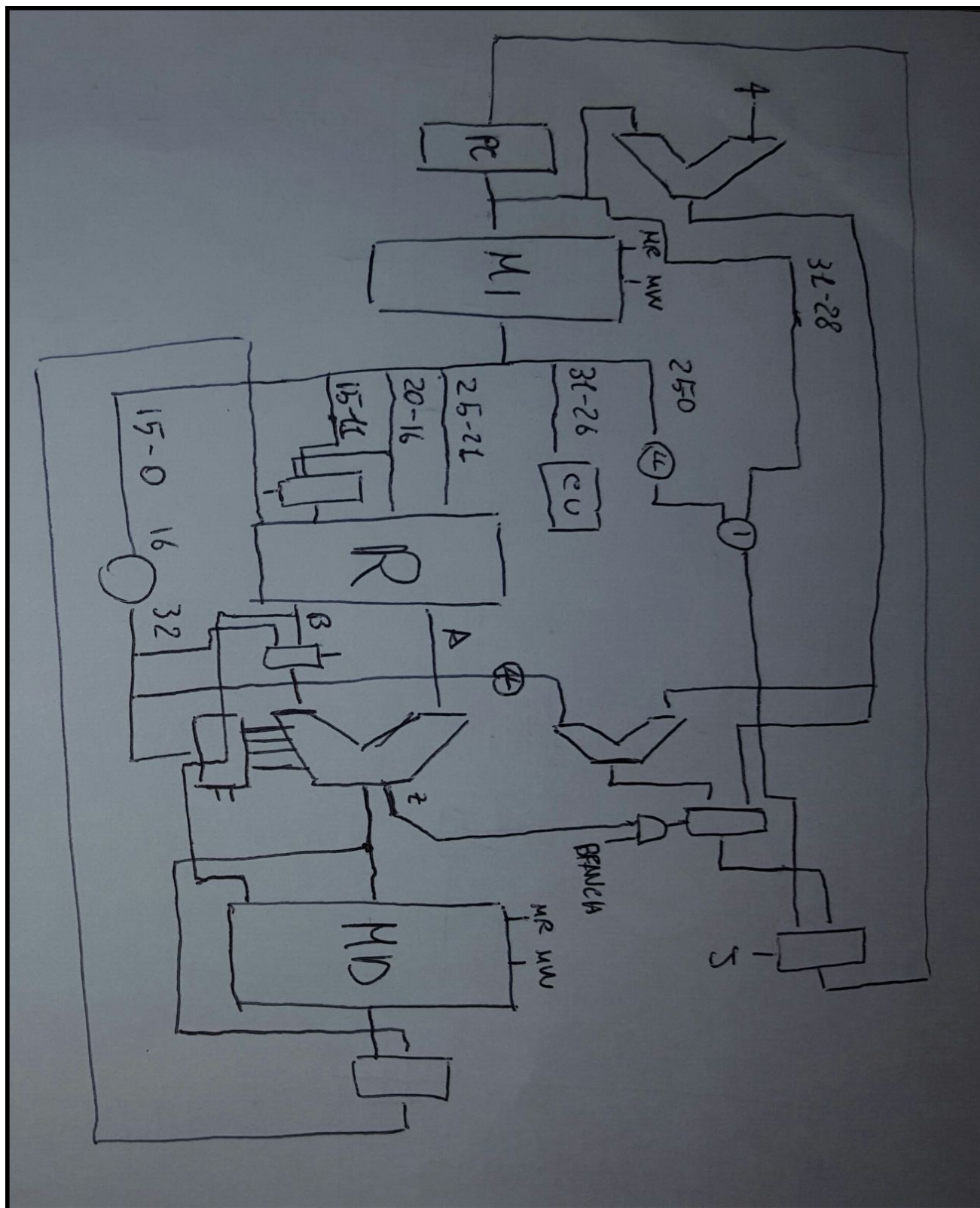
Per quanto riguarda il salto condizionato, in fase FETCH calcoliamo il valore di $PC + 4$; tale valore in fase EXE sarà sommato al campo COSTANTE esteso a 32 e scalato di 2 (questo viene effettuato per rendere quel numero multiplo di 4), il calcolo di questa somma è affidato ad un sommatore apposito e quindi non utilizza l'ALU. Tale sequenza di bit sarà l'indirizzo a cui potenzialmente un'istruzione di salto condizionato vuole saltare. Per decidere se saltare o meno si utilizzerà un multiplexer che prenderà in input il valore di PC e del risultato del sommatore del salto condizionato; come bit di controllo utilizzerà l'AND tra il bit Z prodotto dall'ALU e il bit di BRANCH: bit di controllo che sarà uno solo nel caso di una BEQ.

Per quanto riguarda il salto incondizionato, in fase FETCH prendiamo il valore di PC, in particolare i 4 bit più significativi e li allineiamo ai bit 25-0 dell'istruzione scalati di 2. Visto che abbiamo a disposizione solo 26 bit si è deciso di dividere concettualmente la memoria in 16 blocchi di 2^{28} locazioni, quindi scalando di 2 i 26 bit possiamo

individuare uno tra quelle locazioni. I blocchi sono individuato proprio da i 4 bit più significativi dell'indirizzo dell'istruzione (Nota infatti $16 = 2^4$, $2^4 * 2^{28} = 2^{32}$) . Quindi la combinazione a 32 bit ottenuta come spiegato rappresenta l'indirizzo a cui potenzialmente un'istruzione di salto incondizionato (J) vuole saltare viene utilizzata in fase exe da un secondo multiplexer che prende in input il risultato del multiplexer del salto condizionato e come secondo input l'indirizzo del salto incondizionato , tale multiplexer è controllato dal bit di controllo J, che sarà 1 solo nel caso di un'operazione di salto incondizionato, il risultato di questo multiplexer sarà mandato in input al PROGRAM COUNTER.

Nel caso si abbia la necessità di introdurre la BNE si ha la necessità del secondo bit di branch che andrà in and con il bit z dell'alu negato , il prodotto di tale porta and sarà input di una seconda porta or che prenderà come secondo input l'and tra il primo bit di branch e il bit z, il risultato di tale porta or sarà bit di controllo del multiplexer del salto condizionato.

4.9 PROGETTO FINALE MIPS CICLO SINGOLO



5. PROGETTAZIONE MIPS CON STRUTTURA PIPELINE

5.1 INTRODUZIONE

Con il MIPS a ciclo singolo ogni istruzione ha un numero di fasi pari al numero di dispositivi che l'istruzione utilizza, questo è dato dal fatto che il processore esegue un'istruzione alla volta e quindi in ogni regione del processore ci sarà la stessa istruzione. Se un'istruzione si trova in una certa fase ed utilizza un determinato dispositivo dedicato, gli hardware dedicati alle altre fasi rimangono inutilizzati.

5.1.1 INTRODUZIONE ALLA PIPELINE

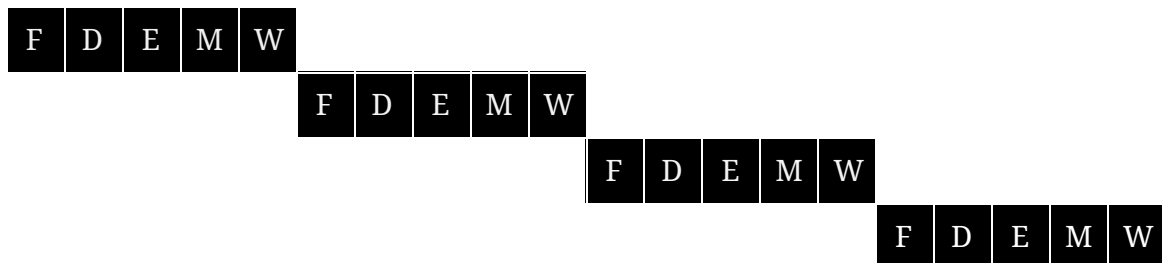
Il ciclo singolo è sicuramente efficiente ma si è cercato nel tempo di ottimizzare le potenzialità del processore introducendo una nuova tecnica di esecuzione delle istruzioni: la "Pipeline". Come visto precedentemente quando un'istruzione utilizza un dispositivo nel ciclo singolo non impiega gli altri, la Pipeline invece cerca di sovrapporle creando un parallelismo in cui ogni dispositivo esegue un'istruzione distinta. Viene quindi diviso il processore in 5 fasi, questa divisione è praticata attraverso dei registri dati chiamati registri di Pipe, ogni fase ha durata uguale alle altre ed esse iniziano/terminano nello stesso momento. Andando da sinistra a destra nel processore vediamo che le istruzioni precedenti occupano la parte destra di quest'ultimo. In fase FETCH avremo quindi l'istruzione più recente, questo causa spesso confusione. Nella Pipeline l'istruzione man mano che attraversa i vari dispositivi del processore si porta con sé i vari bit caratteristici proprio perché il dispositivo che ha appena finito di utilizzare servirà per l'istruzione successiva, se non portasse con sé le informazioni necessarie queste andrebbero perse. Partendo da una Pipe vuota si ha bisogno di 4 cicli completi affinché il processore sia a regime; per regime intendiamo la particolare condizione del processore in cui ad ogni fase successiva alle prime 4 corrisponde un'istruzione terminata.

5.1.2 CALCOLO DELLE PRESTAZIONI

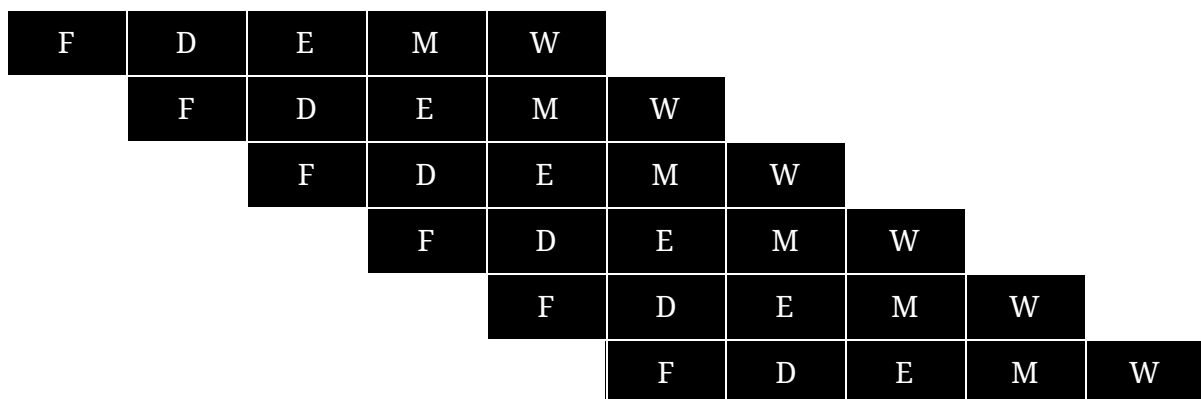
Dal ciclo singolo alla pipe si ha un miglioramento delle prestazioni del processore, poichè non si ha più hardware inutilizzato e quindi il processore viene sfruttato al massimo.

In particolare, in precedenza, il calcolo delle prestazione veniva effettuato sommando i cicli di clock che le singole istruzioni utilizzavano; Nella Pipeline invece bisogna sommare il numero delle istruzione a 4 che sono i cicli per mettere il processore a regime.

CICLO SINGOLO



PIPELINE



**** Una bolla è un istruzione vuota anche detta NO-OP (No operation)**

5.1.3 CRITICITÀ STRUTTURALI

Sono legate al fatto che abbiamo la necessità in ogni caso di tenere la memoria duplicata e dobbiamo avere tre sommatori e un ALU perchè non possiamo avere un unico dispositivo che fa tutti i calcoli perchè alcuni di questi calcoli sono fatti in parallelo da istruzioni diverse.

5.1.4 CRITICITÀ SUI DATI

Le criticità sui dati si verificano quando l'istruzione successiva ha necessità del risultato dell'istruzione subito precedente e non è stato ancora prodotto dalla fase di Write-Back, quindi possono verificarsi solo sulle istruzioni di tipo R, cioè le istruzioni aritmetiche, e per le LW che estrapolano dati dalla memoria.

Le istruzioni che non hanno fase Write-Back non producono dati e quindi non possono produrre criticità sui dati.

Avendo necessità di effettuare dati con i calcoli aggiornati e quindi non lavorare con dati inconsistenti

Senza unità integrate, vengono inserite bolle perchè con la pipeline dobbiamo sovrapporre le fasi Dec e le fasi Write-Back, e dobbiamo effettuare calcoli con i dati aggiornati. Vengono inserite queste “bolle” appunto per conciliare le due fasi.

Se c'è un conflitto dopo ogni istruzione vengono inserite 2 bolle, se c'è un'istruzione in mezzo a queste senza conflitto ne viene inserita solamente 1, mentre con le istruzioni BEQ e BNE vengono introdotte 3 bolle per far sì che sia eseguita l'istruzione corretta.

5.1.5 CRITICITÀ SUL CONTROLLO

Supponiamo che ad un certo punto venga caricata una jump, quindi incomincia la fase fetch di una jump, le istruzioni precedenti sono una in fase dec, una in fase exe, una in fase mem, una in fase write-back, la jump viene caricata, facciamo un passo in avanti e passa in fase dec e ovviamente in fase fetch viene caricata l'istruzione successiva, ma essendo una jump l'istruzione successiva non deve essere eseguita, poiché solo alla fine della fase dec sappiamo che è una jump per cui viene fatto un ulteriore passo in avanti e viene caricata anche la seconda istruzione dopo la jump e solo allora mi accorgo che è una jump, devo aggiornare il PC e devo buttare le due istruzioni caricate.

Invece capisco che è una BEQ sempre alla fine della fase dec, mentre vedo se saltare o meno solo alla fine della fase exe perchè devo controllare il bit 0, quindi carico 3 istruzioni NO-OP dopo la BEQ che bisogna buttare dopo che abbiamo capito che bisogna saltare.

5.2 INTRODUZIONE DELL'UNITÀ DI PROPAGAZIONE

Abbiamo notato che l'utilizzo della Pipeline comporta più di qualche conflitto, ad esempio quando un'istruzione vuole utilizzare un dato prodotto dall'istruzione precedente o da quella a distanza 2. Questo conflitto viene ottimizzato in fase EXE che è anche l'unica fase in cui vengono manipolati bit. Viene introdotto un particolare circuito combinatorio posizionato sotto l'ALU e vengono introdotti due particolari multiplexer il cui output entra in input dell'ALU agli ingressi A e B.

Questo particolare circuito combinatorio, chiamato Unità di Propagazione, produce i bit di controllo dei due multiplexer che da ora in poi chiameremo Propaga-A e Propaga-B.

L'U.P. prende in input 6 bit (2 per istruzione):

1. M/W.rw (Reg-Write) dell'istruzione che si trova in fase Write-Back.
2. E/M.rw (Reg-Write) dell'istruzione che si trova in fase MEM.
3. D (registro destinazione) dell'istruzione che si trova in fase Write-Back.
4. D (registro destinazione) dell'istruzione che si trova in fase MEM.
5. RS dell'istruzione corrente(fase EXE).
6. RT dell'istruzione corrente (fase EXE).

I primi due bit che abbiamo elencato servono affinché non vengano poi fatti confronti inutili. Questi due bit indicano se le istruzioni precedenti abbiano o meno fase di riscrittura nel banco dei registri. Se ciò è vero allora c'è bisogno di fare delle verifiche.

I multiplexer P.A. e P.B. hanno 2 bit di controllo e 3 bit di input.

All'ingresso 0 ricevono rispettivamente A e B dal banco dei registri.

All'ingresso 1 ricevono rispettivamente ALU-OUT dell'istruzione subito precedente ovvero quella che si trova nella fase subito successiva, in questo caso nella fase MEM.

All'ingresso 2 ricevono l'output del multiplexer MEM-to-REG dell'istruzione a distanza 2 che è in fase Write-Back.

L'U.P. vede se M/W.rw è uguale 1, se ciò è vero verifica se RS/RT siano uguali al registro D dell'istruzione in fase WB.

1. RS coincide con D, l'U.P. produce 10 per Propaga-A.

2. RT coincide con D, l'U.P. produce 10 per Propaga-B.

L'U.P. vede se E/M.rw è uguale 1, se ciò è vero verifica se RS/RT siano uguali al registro D dell'istruzione in fase MEM.

1. RS coincide con D, l'U.P. produce 01 per Propaga-A.
2. RT coincide con D, l'U.P. produce 01 per Propaga-B.

Il confronto viene fatto prima con l'istruzione più lontana e poi con quella più vicina poiché l'istruzione precedente potrebbe aver prodotto dati aggiornati rispetto a quella a distanza 2.

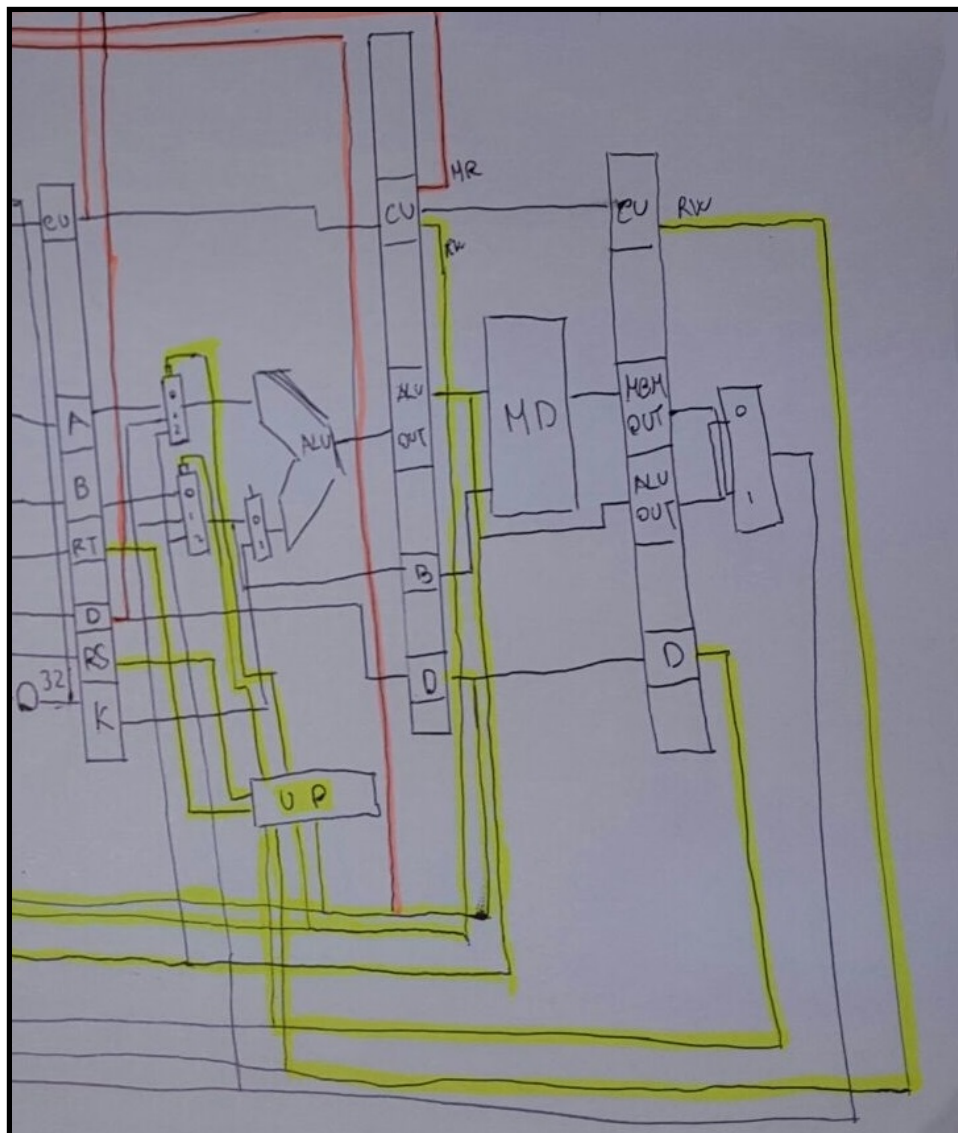
PSEUDO-CODICE

Vedremo qui qual è il codice secondo cui funziona l'unità di propagazione:

```
if (M/W.rw==1) {           /*verifico che l'istruzione più lontana abbia fase di WB*/
    if(D/E.rs==M/W.D)
        P.A.=10;
    if(D/E.rt==M/W.D)
        P.B.=10;
}

if(E/M.rw==1) {           /*verifico che l'istruzione più vicina abbia fase di WB*/
    if(D/E.rs==E/M.D)
        P.A.=01;
    if(D/E.rt==E/M.D)
        P.B.=01;
}
```

Si evince qui come l'U.P. verifichi prima l'istruzione più lontana e poi quella più vicina.



5.3 INTRODUZIONE DELL'UNITÀ DI INDIVIDUAZIONE DEL CONFLITTO

L'U.P. risolve molti problemi, se però abbiamo una LW e l'istruzione successiva vuole utilizzare il dato scaricato dalla memoria ciò non è possibile senza l'integrazione da parte dell'uomo di una bolla. L'Unità di Individuazione del conflitto cerca di risolvere questo problema introducendo autonomamente una NO-OP e cioè producendo 9 bit di controllo a 0 sostituendo quelli della C.U., operando in fase DEC.

L'U.I. viene collocata sopra la C.U. e ha un bit di abilitazione: prende Mem-Read dell'istruzione precedente per vedere se questa sia una LW. Quest'unità controlla 3 diversi dispositivi:

1. Un particolare multiplexer 2 a 1 che prende in input all'ingresso 0 i bit di controllo dalla C.U. e in input all'ingresso 1 9 bit di controllo a 0. Produce gli effettivi bit di controllo per l'istruzione, quando produce i 9 bit a 0 crea una NO-OP, cioè un'istruzione che non produce effetti.
2. Il registro di Pipe F/E, quando viene prodotta una NO-OP la U.I. mette a 0 il clock di questo registro di Pipe affinché i bit dell'istruzione successiva non vengano persi per poterli utilizzare al ciclo successivo.
3. Il P.C., quando viene prodotta una NO-OP la U.I. mette a 0 il clock affinché non venga aggiornato il contatore delle istruzioni per poter rieseguire l'istruzione sostituita dalla NO-OP.

L'U.I. confronta RS e RT dell'istruzione in fase DEC con il campo RT dell'istruzione precedente che in caso di una LW è proprio il registro destinazione ed agisce di conseguenza.

PSEUDO-CODICE

Vedremo qui qual è il codice secondo cui funziona l'unità di individuazione del conflitto:

```
if(F/D.mr == 1) {  
    if(RS == D/E.D || RT == D/E.D) {  
        BUBBLE = 1;  
    }  
}
```

DISEGNO

5.4 ANTICIPAZIONE DELLA BEQ

Come abbiamo visto nel capitolo 5.1.5 ogni istruzione di salto prevede 3 bolle per dare il tempo all'istruzione di salto di aggiornare il program counter se necessario.

Quindi si è cercato di anticipare la beq per diminuire il numero di bolle necessarie, in questo progetto si è deciso di anticipare la beq in fase dec.

La fase dec è divisa in due parti la prima dedicata alla scrittura la seconda dedicata alla lettura. Alla fine della prima fase sono già pronti i bit di controllo e quindi è già possibile capire se l'istruzione in fase dec è un'istruzione di salto o meno. Alla fine della seconda parte abbiamo i dati A e B e dobbiamo produrre il bit zero, nell'ALU lo calcolavamo facendo la sub tra A e B e facendo la NOR di tutti i bit, adesso non avendo l'ALU in fase dec introduciamo 32 porte XOR per effettuare l'or esclusivo bit a bit, e successivamente facciamo un NOR complessivo di tutti e 32 i bit. Quindi prendendo il bit zero prodotto e mettendolo in AND con il bit BRANCH del vettore di controllo sappiamo se bisogna saltare o meno, questo bit sarà il bit di controllo del multiplexer collegato direttamente al program counter. Per calcolare l'indirizzo di salto viene spostato il sommatore prima presente in fase EXE, che prende in input il campo costante a 16 bit estesi a 32 scalati a sinistra di due e PC+4, tale risultato tornerà indietro come secondo ingresso del multiplexer. Come abbiamo visto in precedenza la BEQ può essere soggetta a propagazione, avvenendo in fase DEC non dobbiamo preoccuparci di propagare dalla fase di WRITEBACK poichè scriverà prima che i dati A e B siano letti. I problemi che sorgono sono le istruzioni aritmetiche e le lw che si trovano in fase mem. Per quanto riguarda le istruzioni aritmetiche l'unità di individuazione del conflitto aggiunge una bolla se i registri destinazione o sorgente della beq coincidono con quelli dell'istruzione aritmetica in fase EXE, poichè solo all'inizio della fase MEM avremmo il valore calcolato dall'ALU. Viene aggiunta una seconda unità di propagazione che propaga il risultato dell'ALU dalla fase MEM, tale unità prende in input RS e RT della beq, il registro destinazione dalla fase MEM, il bit reg write per vedere se si tratta di un'istruzione che ha la fase di write back e i registri RS e RT. Tale unità di propagazione propaga solo nel caso RS e RT siano uguali a D. I bit prodotti da tale unità saranno i bit di controllo di due multiplexer 2 a 1 che prenderanno in input A e B prodotti dal banco dei registri al primo ingresso e il valore prodotto dall'ALU dal registro EXE / MEM al secondo ingresso. L'unico problema che rimane risolvere è quello delle LW che si trovano in fase MEM poichè solo alla fine di tale fase abbiamo il valore prelevato dalla memoria, quindi bisogna aspettare che la memoria dati produca il dato e non può essere propagato prima di allora; per tale motivo l'unità di individuazione dei conflitti nel caso di una BEQ se RS e RT coincidono con il campo destinazione del registro EXE / MEM e si tratta di una LW e quindi MR è a 1 aggiunge una bolla.

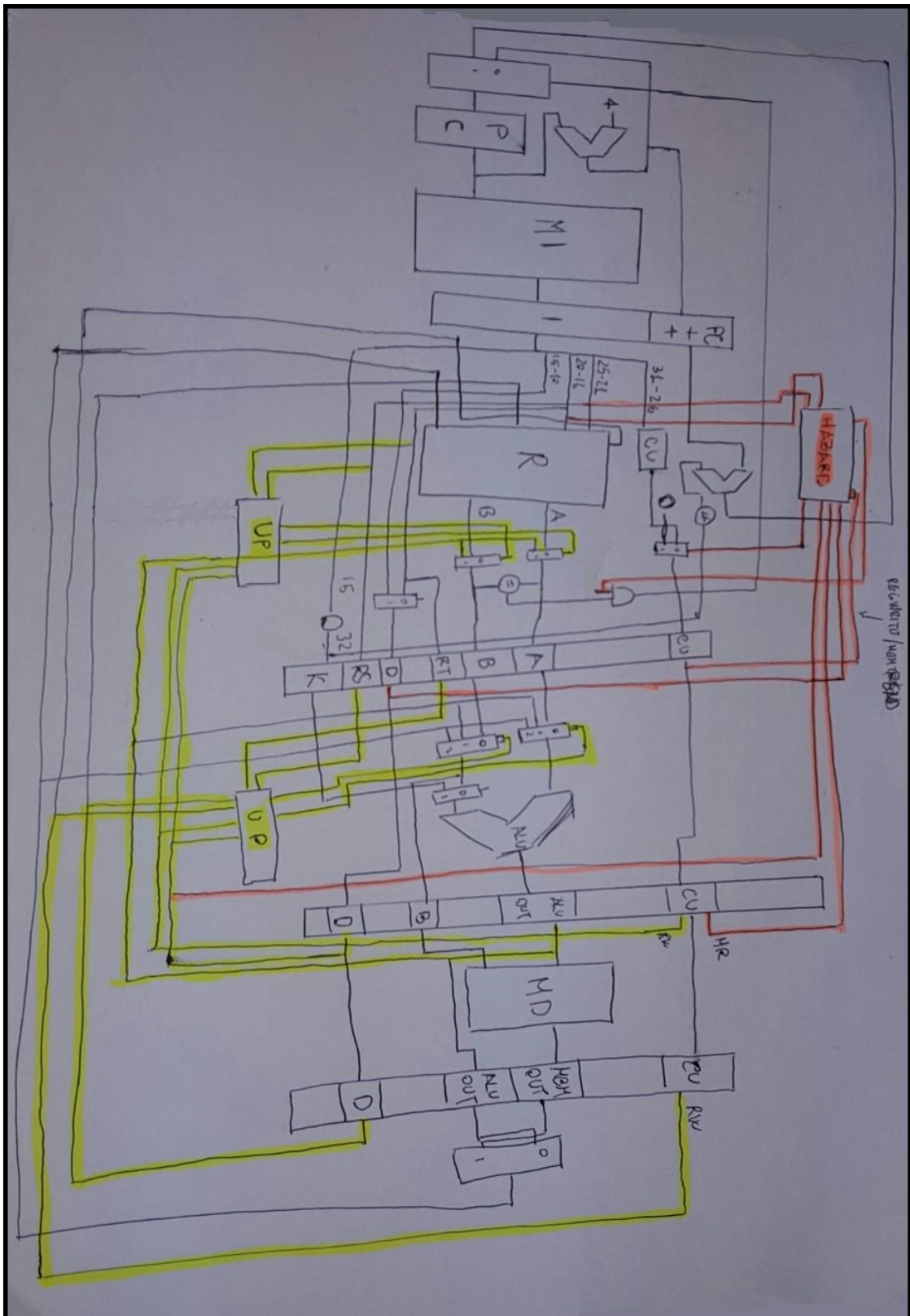
Ogni qual volta la BEQ salta viene sempre aggiunta una bolla mettendo a 0 tutta l'istruzione che si trova nel registro FETCH / DEC.

PSEUDO-CODICE

Vedremo qui qual è il codice secondo cui funziona l'unità di individuazione del conflitto con l'anticipo della beq:

```
if((RS == D/E.D || RT == D/E.D) && (D/E.RW == 1)) {  
    if(D/E.MR == 1 || BRANCH == 1) {  
        BUBBLE = 1;  
    }  
} else if((RS == E/M.D || RT == E/M.D) && (E/M.MR == 1) && BRANCH == 1) {  
    BUBBLE = 1;  
}
```

5.5 PROGETTO FINALE MIPS CON STRUTTURA PIPELINE



6. GERARCHIA DI MEMORIA

6.1 PRIMA PANORAMICA

Con l'evoluzione dei programmi che via via hanno avuto sempre di più la necessità di grande quantità di memoria sia per il numero di istruzioni sia per la quantità di dati che esse utilizzavano era impensabile avere un solo livello di memoria collegato direttamente al processore, quindi si sono arrivati ad avere fondamentalmente 3 strati di memoria : la cache, la memoria centrale e la memoria di massa. Ogni memoria è immagine di una parte della memoria di strato inferiore e quindi di grandezza maggiore.

La cache è l'unica memoria a cui il processore ha accesso, è di piccole dimensioni rispetto alla memoria centrale e alla memoria di massa e ha la particolarità di essere fisicamente divisa in due memorie cache dati e cache istruzioni (che sono quelle fino ad ora citate come memoria dati e memoria istruzione). La cache è strettamente legata alla memoria centrale , infatti è immagine di una parte della memoria centrale, quindi c'è un forte legame tra gli indirizzi della cache e quella della memoria.

MEMORIA CACHE

MEMORIA RAM

MEMORIA
DI MASSA

6.2 CRITERI DI GESTIONE

Le uniche istruzioni che possono essere eseguite dal processore sono quelle presenti nella memoria cache, tali istruzioni vengono caricate dalla memoria centrale. Le politiche con cui vengono caricate le istruzioni dalla memoria centrale alla cache sono di due tipi : di località temporale e di località spaziale.

Le politiche di località temporali prevedono il caricamento dell'istruzione in base all'esecuzione, quindi si prevede che le istruzioni a cui si è fatto riferimento vengano richieste nuovamente in tempi brevi.

Le politiche di località spaziale prevedono il caricamento delle istruzioni in blocco e scelgo le istruzioni con indirizzi di memoria vicini.

6.3 CORRISPONDENZA TRA INDIRIZZI IN MEMORIA E IN CACHE

Dato lo stretto legame tra cache e memoria centrale si ha la necessità di trovare il modo di creare una corrispondenza tra indirizzi della memoria centrale e indirizzi della cache , esistono diversi tipi di corrispondenza :

Corrispondenza Diretta

Tale corrispondenza prevede uno stretto legame tra gli indirizzi della cache e quelli della memoria centrale , dove una parte dei bit sono usati per identificare l'indirizzo nella cache, quindi ad un indirizzo della cache verrà associato un set di indirizzi della memoria centrale, per tale motivo se la cache avrà N celle $\log_2(N)$ è il numero di bit necessari affinché si possano indicizzare tutte le celle.

Esempio : Cache con indirizzi a 3 bit e Memoria Centrale con indirizzi a 5bit

La Cache avrà 2^3 celle quindi 8 locazione, mentre la memoria centrale ne avrà 2^5 locazioni quindi 32. Dei 5 bit dedicati ad un indirizzo della memoria centrale $\log_2(8)$ saranno utilizzati per indicare una specifica locazione della cache. Quindi ad ogni locazione della cache saranno associati 4 locazioni della memoria centrale, ne seguono esempi.

ADDR Memoria Centrale 11011 => ADDR Cache corrispondente 011

ADDR Cache 101 => ADDR Memoria Centrale corrispondenti :

00101 - 01101 - 10101 - 11101.

Per effettuare la corrispondenza diretta come detto bisogna quindi dividere l'indirizzo della memoria centrale, la parte dedicata all'indicizzazione della cache è chiamata INDICE mentre la parte restante che individua a quale cella della memoria

centrale facciamo riferimento è chiamata TAG.

Quindi la cache ogni locazione della cache avrà i 32 bit della parola (il contenuto) e il TAG che indica a quale cella della memoria centrale corrisponde di quelle che sono associate a quella locazione della cache. C'è anche un ulteriore campo il bit di validità che se è 0 il significa che la locazione è vuota, in caso contrario vale 1.

Quando tentiamo l'accesso alla memoria , l'indirizzo contenuto nel program counter fa riferimento alla memoria centrale e non alla cache. Tale indirizzo mediante la corrispondenza diretta come descritto viene diviso, una parte serve per indicizzare la cache e tale parte verrà inviata alla memoria istruzione, che attiverà la lettura di quella specifica locazione, oltre a leggere il contenuto verrà letto anche il tag e il bit di validità.

I rimanenti bit dell'indirizzo contenuti nel PC saranno inviati in input ad un comparatore insieme al tag estratto dalla cache, il risultato di tale comparatore sarà messo in AND con il bit di validità se il risultato di questa porta AND sarà uno allora il dato è valido.

