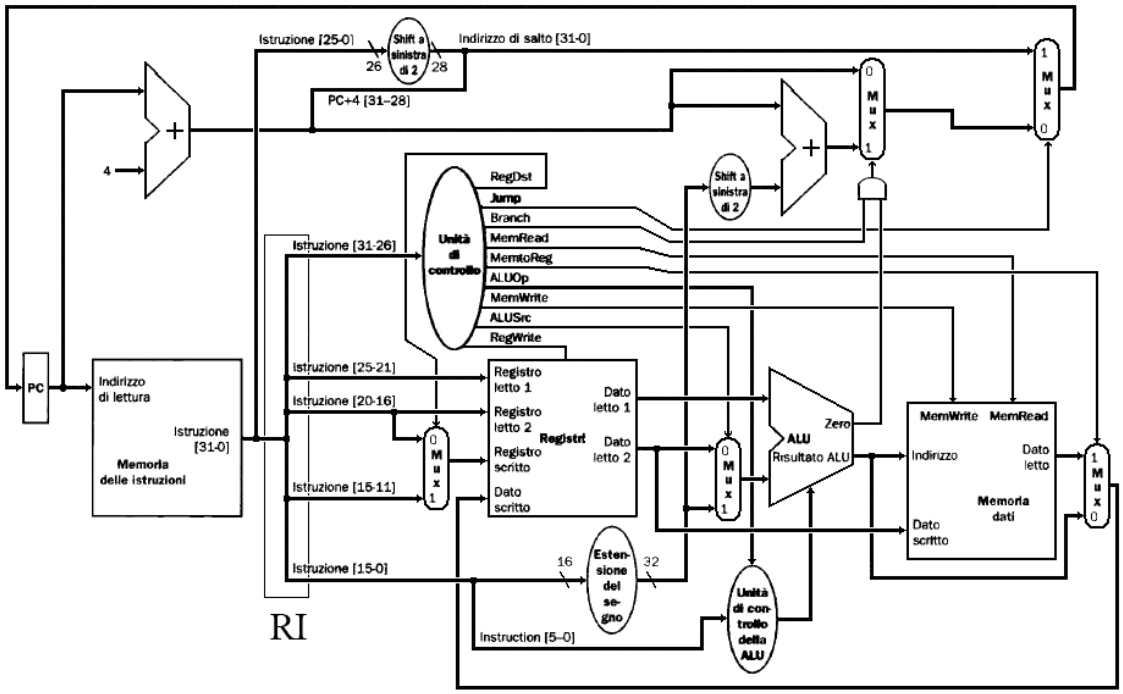
Esercitazione del 14/11/2011 – Soluzioni

1. CPU singolo ciclo

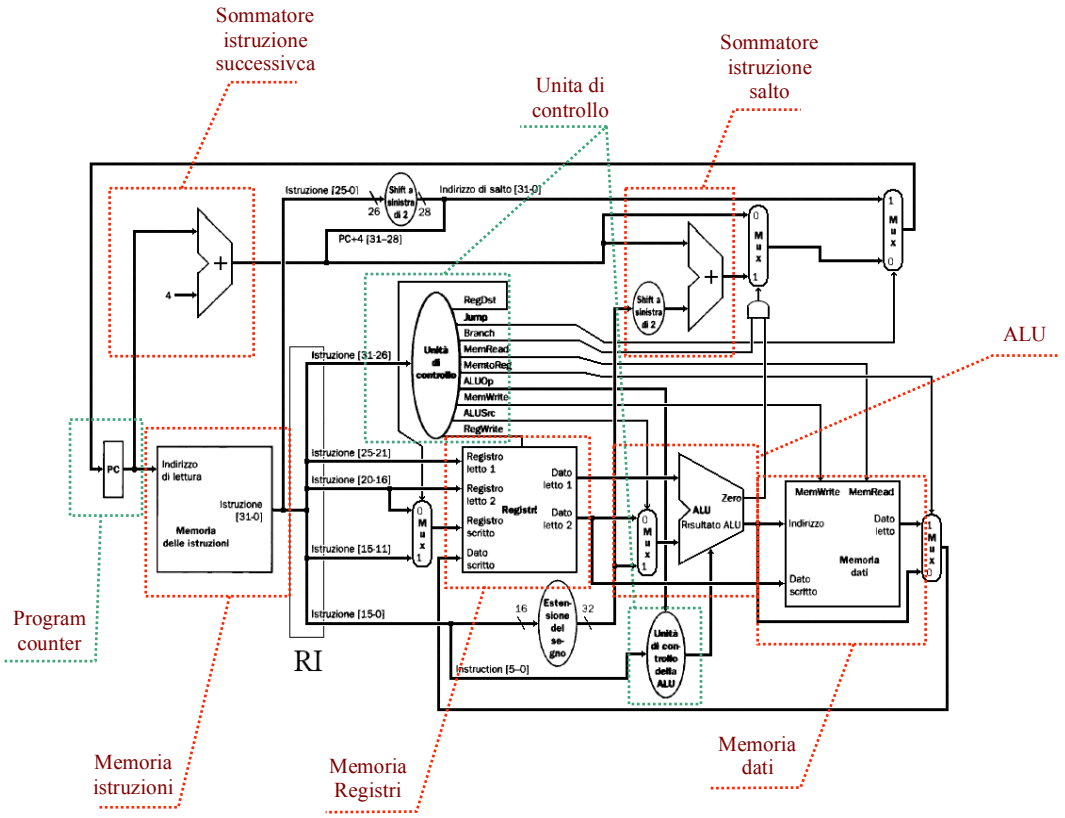
Una CPU singolo ciclo è una CPU che riesce a computare ogni istruzione sfruttando un solo ciclo di clock. All'atto pratico può essere vista come una rete sequenziale in cui la memoria è costituita dalla memoria dati e codice, e dai registri. Le transizioni da uno stato all'altro avvengono all'interno di un singolo ciclo di clock. Il codice operativo dell'istruzione viene analizzato dalla rete di controllo che costruisce, per ogni classe di istruzioni, il percorso circuitale tra i vari blocchi che computa l'istruzione stessa.
All'interno del ciclo di clock la CPU si comporta come una rete combinatoria: il risultato viene ottenuto costruendo percorsi di collegamento opportuni e propagando i dati di ingresso fino alle uscite della rete. Al successivo colpo di clock i risultati vengono memorizzati nella memoria e/o nei registri.

Di seguito è riportata una CPU a singolo ciclo di clock semplificata che implementa solo le classi di istruzione **R**, **J**, **branch** e **Load/Store**.:



Massimo Marchi – marchi@dsi.unimi.it

All'interno della CPU a singolo ciclo si possono evidenziare i seguenti blocchi:



- Sommatore per il calcolo dell'indirizzo dell'istruzione successiva
- Sommatore per il calcolo dell'indirizzo dell'istruzione puntata da un eventuale branch.
- Program Counter
- Memoria testo
- Unita di controllo
- Memoria registri
- ALU
- Memoria dati

A causa della sua architettura, sulla CPU a singolo ciclo di clock si possono notare alcune caratteristiche:

- sono presenti più moduli simili che eseguono la stessa operazione ma su dati diversi, questo comporta una maggiore complessità circuitale (ALU e sommatori, memorie dati e testo)
- sono presenti due memorie distinte, una per l'area dati ed una per l'area codice.

La necessita di avere due memorie distinte viene dal fatto che durante lo stesso ciclo di clock un'istruzione può richiedere di eseguire più accessi contemporanei alla memoria,

ad esempio, il fetch dell'istruzione da eseguire e contemporaneamente una scrittura su registri; anche se le due operazioni sono eseguite in tempi diversi, i corrispondenti circuiti restano impegnati per tutto il ciclo di clock.

Supponiamo ora che alla posizione **0x0040.0000** sia presente l'istruzione

ADD $10, $8, $9

L'istruzione è di tipo R, la sua trama quindi è la seguente :

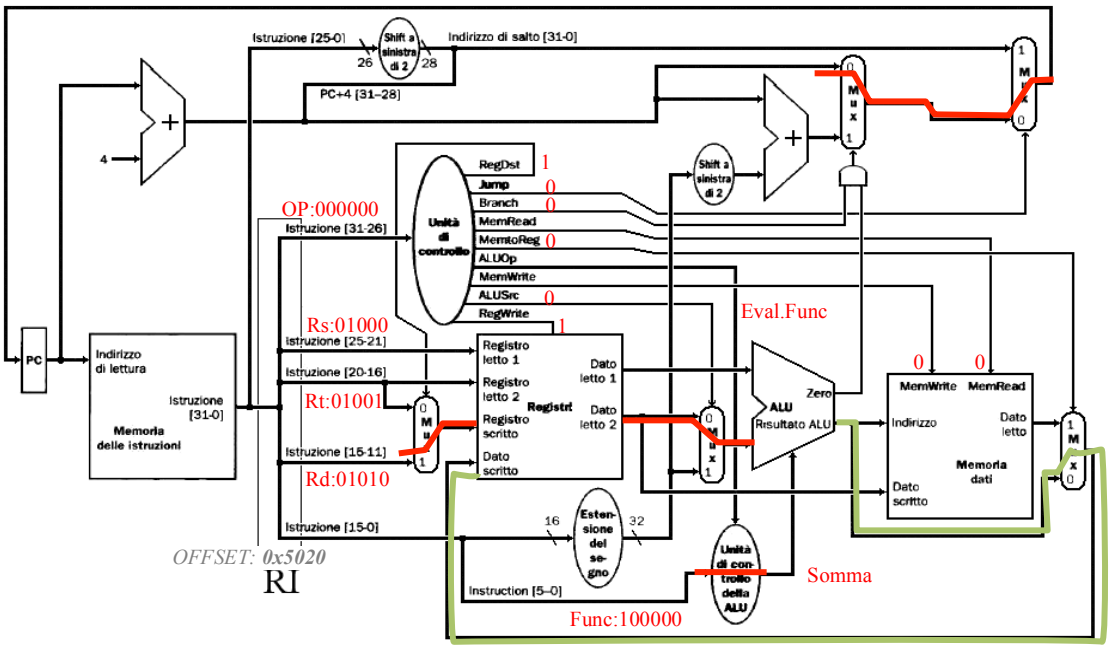
ADD rd, rs, rt

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	**rs**	**rt**	**rd**	00000	**ADD** 100000

Il codice macchina corrispondente è:

0000.00 01.000 0.1001 0101.0 000 00 10.0000 = 0x01095020

Consideriamo lo schema precedente e supponiamo che all'inizio del ciclo di clock il **PC** valga **0x0040.0000:**



Dopo un transitorio sull'bus di uscita della memoria test è presente il codice macchina che viene decodificato dall'unità di controllo come indicato.

Supponiamo ora che alla posizione **0x0040.0000** sia presente l'istruzione

LW $9, 3($8)

L'istruzione è di tipo **Load/Store**, la sua trama quindi è la seguente :

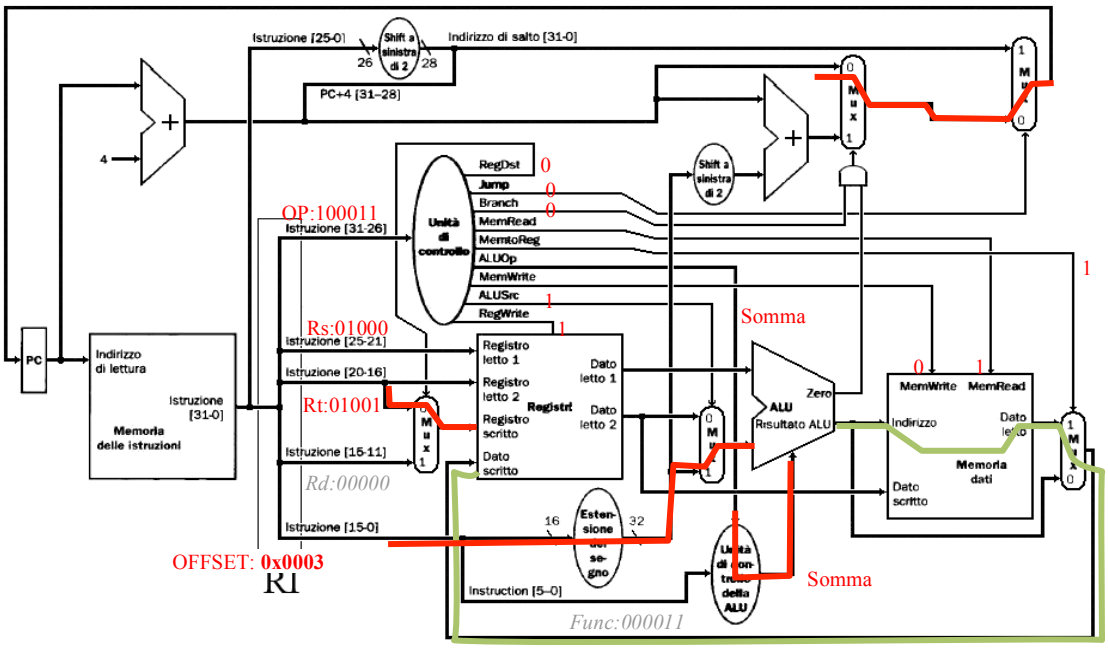
LW rt , offset (rb)

31 26	25 21	20 16	15 0
LW 100011	**base**	**rt**	**offset**

Il codice macchina corrispondente è:

1000.11 01.000 0.1001 0000.0000.0000.0011 = 0x8d090003

Come prima supponiamo che all'inizio del ciclo di clock il **PC** valga **0x0040.0000:**



Supponiamo ora che alla posizione **0x0040.0000** sia presente l'istruzione

SW $9, 3($8)

L'istruzione è di tipo **Load/Store**, la sua trama quindi è la seguente :

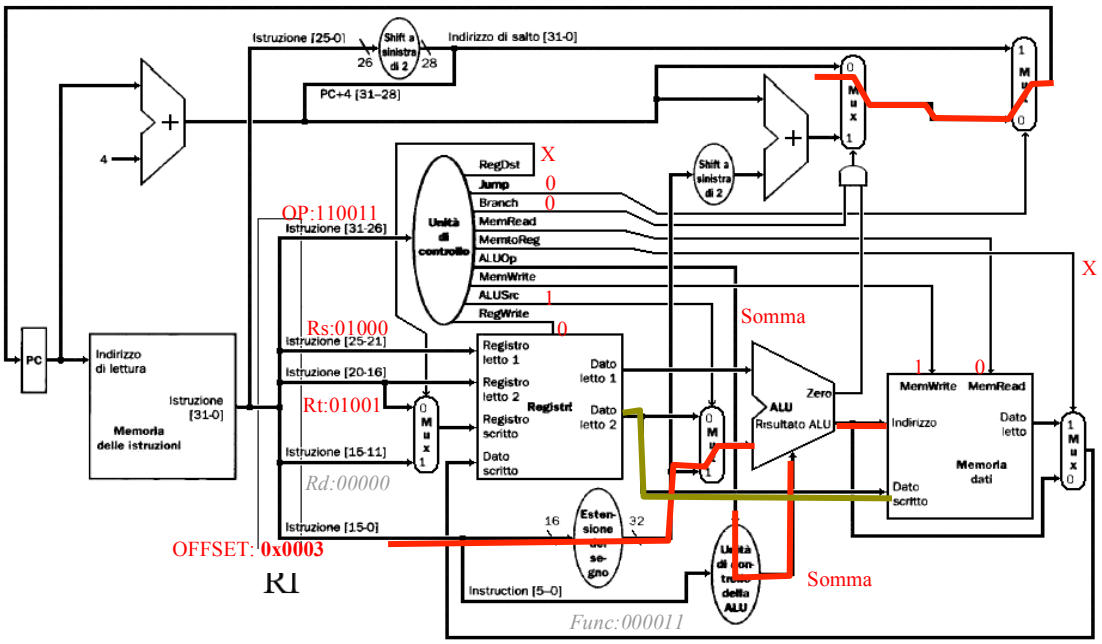
SW rt , offset (rb)

31 26	25 21	20 16	15 0
SW 110011	**base**	**rt**	**offset**

Il codice macchina corrispondente è:

1100.11 01.000 0.1001 0000.0000.0000.0011 = 0xad090003

Come prima supponiamo che all'inizio del ciclo di clock il **PC** valga **0x0040.0000:**



Supponiamo ora che alla posizione **0x0040.0000** sia presente l'istruzione

BEQ $8, $9, 0x0040.0004

L'istruzione è di tipo **Brach**, la sua trama quindi è la seguente :

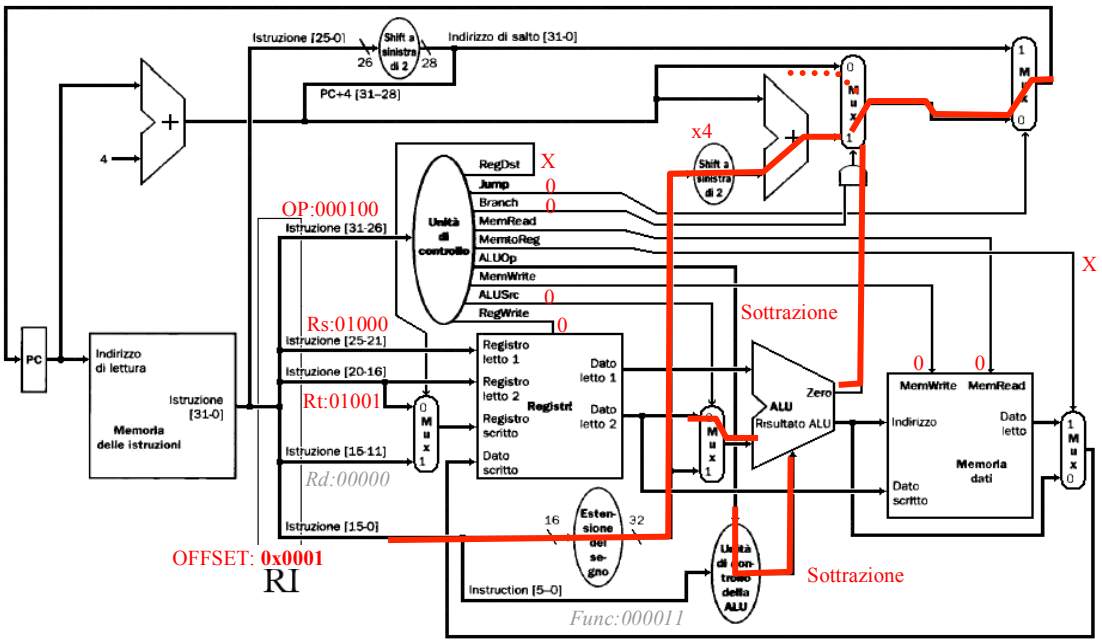
BEQ rs , rt, offset

31 26	25 21	20 16	15 0
BEQ 000100	**rs**	**rt**	**offset**

Il codice macchina corrispondente è:

0001.00 01.000 0.1001 0000.0000.0000.0001 = 0x11090001

Come prima supponiamo che all'inizio del ciclo di clock il **PC** valga **0x0040.0000:**



Supponiamo ora che alla posizione **0x0040.0000** sia presente l'istruzione

J 0x004000034

L'istruzione è di tipo **Jump**, la sua trama quindi è la seguente :

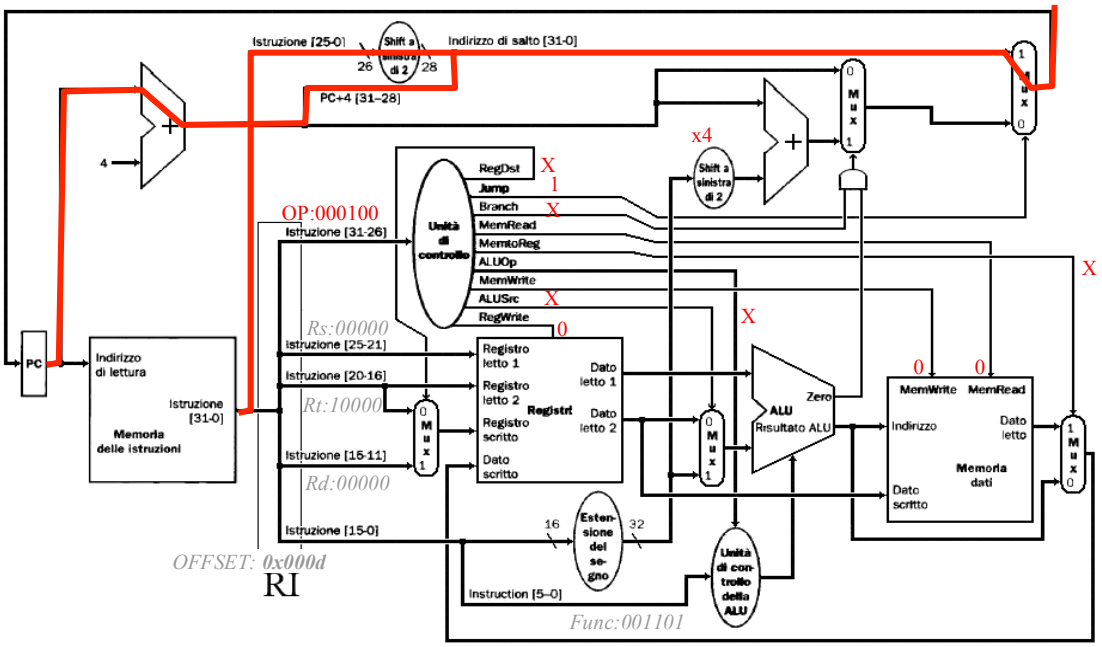
J offset

31 26	25 21	20 16	15 0
J 000010	**rs**	**rt**	**Offset**

Il codice macchina corrispondente è:

0000.10 00.0001.0000.0000.0000.0000.1101 = 0x0810000d

Come prima supponiamo che all'inizio del ciclo di clock il **PC** valga **0x0040.0000:**



Di seguito è riportata la tabella riassuntiva dei segnali di controllo generati dall'unità di controllo:

	OpCode	RegDst	AluSrc	MemToReg	RegWrite	MemRead	MemWrite	Branch	Jump	AluOp
ADD	000.000	1	0	1	1	0	0	0	0	10 (Funct)
LW	100.011	0	1	0	1	1	0	0	0	00 (Add)
SW	110.011	X=0	1	0	0	0	1	0	0	00 (Add)
BEQ	000.100	X=1	0	1	0	0	0	1	0	01 (Sub)
J	000.010	X=1	X=0	X=1	0	0	0	X=0	1	XX=10

Se indichiamo con $Op_5Op_4Op_3Op_2Op_1Op_0$ l'OpCode e con Aop_1 Aop_0 il segnale di AluOp possiamo dare un'implemetazione circuitale dell'unità di controllo. Assegnando in maniera opportuna i segnali indifferenti possiamo ottenere la seguente funzione:

$$RegDst = \sim Op_0$$

$$AluSrc = Op_0$$

$$MemToReg = \sim Op_0$$

Manipolando opportunamente le configurazioni di OpCode non indicate in tabella (aggiungo un 1 per le configurazioni non utilizzate 001000,101011,11X000,11X111) possiamo ottenere:

$$\begin{aligned} RegWrite &= \sim Op_5\sim Op_4\sim Op_3\sim Op_2\sim Op_1\sim Op_0 + Op_5\sim Op_4\sim Op_3\sim Op_2Op_1Op_0 \\ &+ \sim Op_5\sim Op_4Op_3\sim Op_2\sim Op_1\sim Op_0 + Op_5\sim Op_4Op_3\sim Op_2Op_1Op_0 \\ &= \sim Op_4\sim Op_2(\sim Op_5\sim Op_1\sim Op_0 + Op_5Op_1Op_0) \\ &+ Op_4Op_2(\sim Op_5\sim Op_1\sim Op_0 + Op_5Op_1Op_0) \\ &= \sim Op_5\sim Op_1\sim Op_0 + Op_5Op_1Op_0 \\ &= (Op_5Op_1Op_5Op_0 + \sim Op_5\sim Op_1Op_5Op_0 \\ &+ Op_5Op_1\sim Op_5\sim Op_0 + \sim Op_5\sim Op_1\sim Op_5\sim Op_0 \\ &= (Op_5Op_1 + \sim Op_5\sim Op_1)(Op_5Op_0 + \sim Op_5\sim Op_0) \\ &= (\sim Op_5 \oplus Op_1)(\sim Op5 \oplus Op_0) \end{aligned}$$

Considerando i bit Op5 e Op4 possiamo ricavare allo stesso modo le seguenti funzioni:

$$MemRead = Op_5\sim Op_4$$

$$MemWrite = Op_5Op_4$$

Possiamo notare che per la configurazione Jump e solo per quella, Op ed Op_0 sono diversi. Sfruttando la tabella della verità di Xor otteniamo:

$$Branch = Op_2$$

$$Jump = Op_5 \oplus Op_4$$

Infine, AluOp può essere calcolata facilmente notando che la funzione $AluOp_1$ è simole ad una Nor, e che la $AluOp_0$ insegue Op_2:

$$AluOp_1 = \sim(Op5+Op1)$$

$$AluOp_0 = Op_2$$

Massimo Marchi – marchi@dsi.unimi.it