Specifica del Progetto di Laboratorio
Architettura degli Elaboratori I

Alessandro Capezzera*
matricola: 938359, Turno: A
alessandro.capezzera@studenti.unimi.it

1 Introduzione al progetto

Il progetto consiste nella realizzazione di un'architettura RISC a 32bit, prendendo spunto dall'architettura MIPS, con ISA ridotto e un'ALU ad elaborazione di numeri interi. Le istruzioni vengono inserite nell'Istruction Register tramite l'editor di logisim. Una volta attivato il clock, le istruzioni vengono eseguite in cascata fino all'ultima disponibile nell'IR. Al fine di dimostrare il funzionamento viene caricato un programma che inserisce 6 elementi in memoria centrale, da 4 byte ciascuno, sotto forma di array, un elemento k nel registro $a1, scansiona l'array e inserisce nel registro $v0: 1 se è stato trovato l'elemento k, 0 altrimenti.

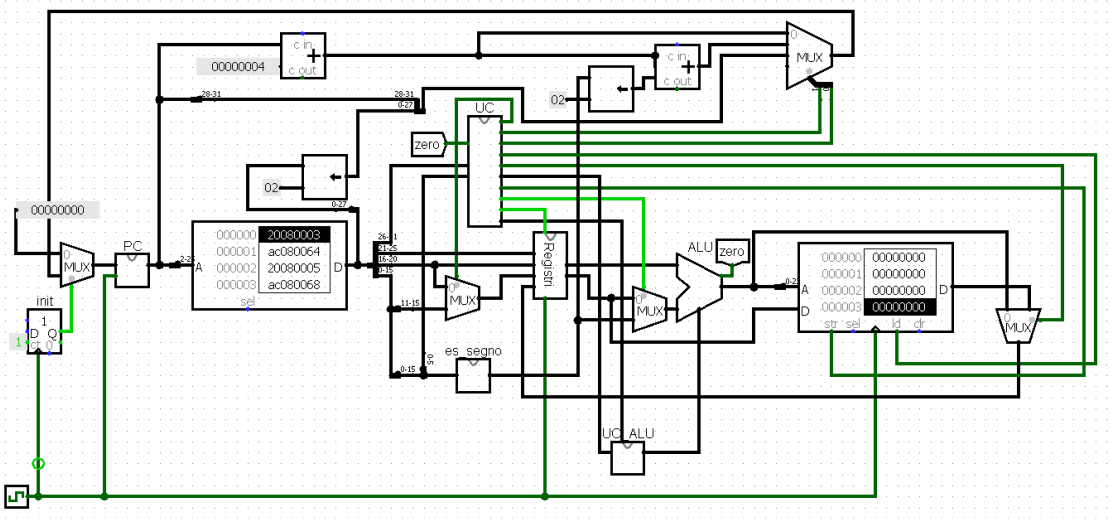


Figura 1: Circuito in esecuzione

*Progetto approvato il 29/04/2019, consegnato il 06/06/2019

1.1 ISA

Tutte le istruzioni, oltre alla loro naturale funzione, fanno avanzare il valore del Program Counter di 4 ad ogni ciclo di esecuzione durante la fase di fetch.
Nota: Tutti i valori aritmetici immediati sono codificati in complemento a 2.
Nota 2: Nelle tabelle che seguono la variabile MEM identifica la memoria centrale come array di byte.

add

Descrizione	Somma due registri e salva il risultato in un registro.
Operazione	$d = $s + $t;
Sintassi	add $d, $s, $t
Codifica	0000 00ss ssst tttt dddd d000 0010 0000

sub

Descrizione	Sottrae due registri e memorizza il risultato in un registro.
Operazione	$d = $s - $t;
Sintassi	sub $d, $s, $t
Codifica	0000 00ss ssst tttt dddd d000 0010 0010

slt

Descrizione	Se $s è minore di $t, $d è impostato a 1. E' impostato a 0 altrimenti.
Operazione	`if($s < $t) $d = 1; else $d = 0;`
Sintassi	slt $d, $s, $t
Codifica	0000 00ss ssst tttt dddd d000 0010 1010

or

Descrizione	Or logico bit a bit tra due registri e memorizza il risultato in un registro.
Operazione	`$d = $s | $t;`
Sintassi	or $d, $s, $t
Codifica	0000 00ss ssst tttt dddd d000 0010 0101

and

Descrizione	And logico bit a bit tra due registri e memorizza il risultato in un registro.
Operazione	$d = $s & $t;
Sintassi	and $d, $s, $t
Codifica	0000 00ss ssst tttt dddd d000 0010 0100

nor

Descrizione	Nor logico bit a bit tra due registri e memorizza il risultato in un registro.
Operazione	`$d = ~($s | $t);`
Sintassi	nor $d, $s, $t
Codifica	0000 00ss ssst tttt dddd d000 0010 0111

addi

Descrizione	Somma un registro e un valore immediato con segno esteso e memorizza il risultato in un registro.
Operazione	$t = $s + imm;
Sintassi	addi $t, $s, imm
Codifica	0010 00ss ssst tttt iiii iiii iiii iiii

slti

Descrizione	Se $s è meno del valore immediato, $t è impostato a 1. E' impostato a 0 altrimenti.
Operazione	`if($s < imm) $t = 1; else $t = 0;`
Sintassi	slti $t, $s, imm
Codifica	0010 10ss ssst tttt iiii iiii iiii iiii

ori

Descrizione	Or logico bit a bit tra un registro e un valore immediato e memorizza il risultato in un registro.
Operazione	`$d = $s | imm;`
Sintassi	ori $t, $s, imm
Codifica	0011 01ss ssst tttt iiii iiii iiii iiii

andi

Descrizione	And logico bit a bit tra un registro e un valore immediato e memorizza il risultato in un registro.
Operazione	$t = $s & imm;
Sintassi	andi $t, $s, imm
Codifica	0011 00ss ssst tttt iiii iiii iiii iiii

j

Descrizione	Salta all'indirizzo calcolato.
Operazione	`PC = (PC & 0xf0000000) | (target << 2);`
Sintassi	j target
Codifica	0000 10ii iiii iiii iiii iiii iiii iiii

beq

Descrizione	Salta se i due registri sono uguali.
Operazione	`if($s == $t) PC = PC + (offset << 2);`
Sintassi	beq $s, $t, offset
Codifica	0001 00ss ssst tttt iiii iiii iiii iiii

bne

Descrizione	Salta se i due registri non sono uguali.
Operazione	`if($s != $t) PC = PC + (offset << 2);`
Sintassi	bne $s, $t, offset
Codifica	0001 01ss ssst tttt iiii iiii iiii iiii

lw

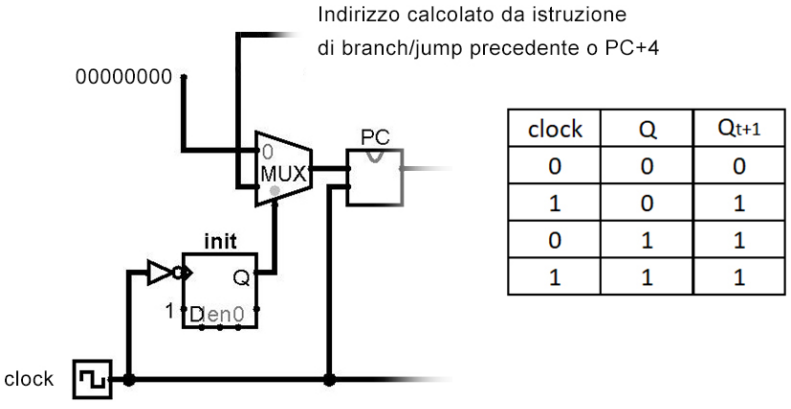
Descrizione	Una word viene caricata in un registro dall'indirizzo specificato.
Operazione	$t = MEM[$s + offset];
Sintassi	lw $t, offset($s)
Codifica	1000 11ss ssst tttt iiii iiii iiii iiii

sw

Descrizione	Il contenuto di $t è memorizzato in memoria all'indirizzo specificato.
Operazione	MEM[$s + offset] = $t;
Sintassi	sw $t, offset($s)
Codifica	1010 11ss ssst tttt iiii iiii iiii iiii

2 Inizializzazione

Abilitando il clock su logisim, nella fase di fetch, l'indirizzo da caricare nel Program Counter viene filtrato da un multiplexer comandato da un flip flop. Il flip flop ha la funzione di caricare: l'indirizzo di partenza da cui iniziare ad eseguire le istruzioni(ad esempio: 0x00000000) al primo ciclo di clock; il contenuto del PC+4 o l'indirizzo calcolato da eventuali istruzioni di branch/jump i successivi cicli.



clock	Q	Q_{t+1}
0	0	0
1	0	1
0	1	1
1	1	1

Figura 2: Componenti della fase di inizializzazione

3 Decodifica

Le istruzioni, lunghe 32bit, per scelta implementativa hanno lo stesso codice operazione, campo funzione e formato delle istruzioni MIPS, nonostante l'ISA ridotto permetta un utilizzo inferiore di bit per l'OpCode.
Capacità ISA con 6bit di OpCode: 2^6-1 istruzioni + 2^6 istruzioni di tipo R con OpCode 0b000000.

3.1 Formato istruzioni

R: Il formato R (register) è utilizzato per istruzioni aritmetiche e logiche. Specifica 3 registri nel suo formato(rs,rt,rd).

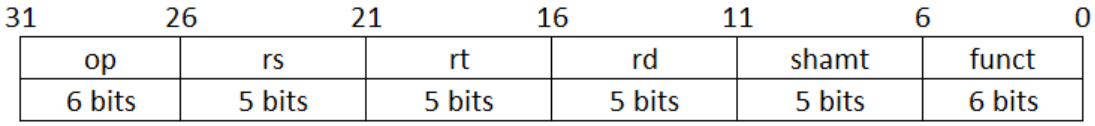


Figura 3: Formato istruzioni di tipo R

I: Il formato I (immediate) è utilizzato per istruzioni di memoria, immediate e di branch. Specifica 2 registri nel suo formato(rs,rt).

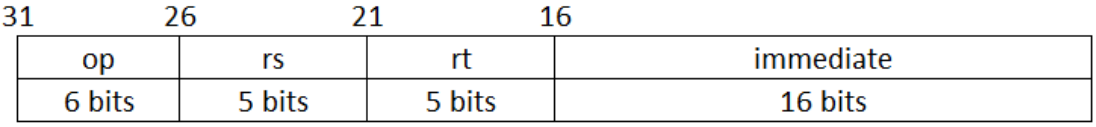


Figura 4: Formato istruzioni di tipo I

J: Il formato J (Jump) è utilizzato per le istruzioni di salto incondizionato. Non specifica alcun registro nel suo formato.

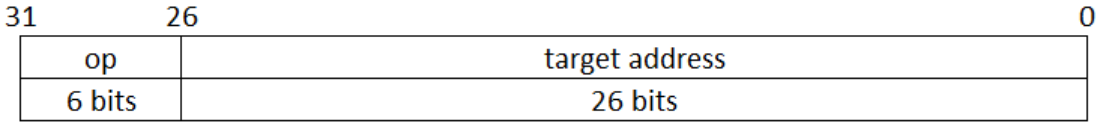


Figura 5: Formato istruzioni di tipo J

3.2 campi

op: Codice operazione dell'istruzione.
rs: Registro sorgente dell'istruzione.
rt: Registro target dell'istruzione.
rd: Registro destinazione dell'istruzione.
shamt: Nelle istruzioni di tipo R indica lo shift amount (solo per le operazioni di shift. Altrimenti ha valore 0b00000, nell'implementazione corrente dell'architettura non sono presenti istruzioni di shift).
funct: Nelle istruzioni di tipo R indica l'operazione da effettuare tra i registri rs e rt.
immediate: Valore immediato per le operazioni aritmetico/logiche o offset dell'indirizzo per le istruzioni di memoria (lw/sw).
target address: Indirizzo target di un'istruzione di jump.

4 Unità di controllo

Il compito dell'unità di controllo è di generare tutti i segnali di controllo che coordinano le operazioni nelle varie parti della CPU. Prende in input l'OpCode dai 6 bit più significativi dell'istruzione e il segnale di zero dell'ALU(calcolato solo in un secondo momento). Restituisce tutti i segnali che servono per il controllo di flusso dell'istruzione(vedi paragrafo 4.1).

	OpCode	RegDst	ALUSrc	Mem2Reg	RegWrite	MemRead	MemWrite	Branch	ALUs	J
add	000000	1	0	0	1	0	0	0	010	0
sub	000000	1	0	0	1	0	0	0	010	0
slt	000000	1	0	0	1	0	0	0	010	0
or	000000	1	0	0	1	0	0	0	010	0
and	000000	1	0	0	1	0	0	0	010	0
nor	000000	1	0	0	1	0	0	0	010	0
addi	001000	0	1	0	1	0	0	0	011	0
slti	001010	0	1	0	1	0	0	0	111	0
ori	001101	0	1	0	1	0	0	0	001	0
andi	001100	0	1	0	1	0	0	0	000	0
j	000010	X	X	X	0	X	0	X	X	1
beq	000100	X	0	X	0	0	0	1	110	0
bne	000101	X	0	X	0	0	0	1	110	0
lw	100011	0	1	1	1	1	0	0	011	0
sw	101011	X	1	X	0	0	1	0	011	0

Figura 6: Tabella di verità unità di controllo

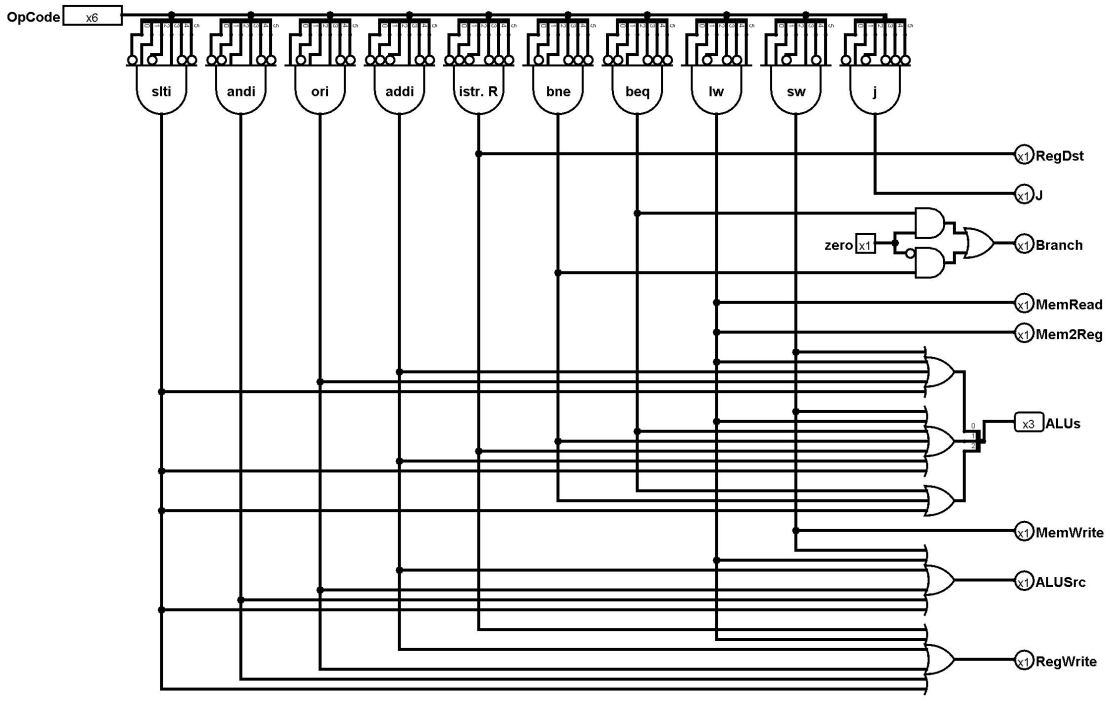


Figura 7: Circuito unità di controllo

4.1 Segnali di controllo

RegDst: Comanda il multiplexer che seleziona i 5 bit d'indirizzo del registro su cui scrivere: rd nel caso delle istruzioni di tipo R (RegDst=1), rt nel caso delle istruzioni di tipo I(RegDst=0).

J: Il segnale di jump, insieme al segnale di branch, comanda il multiplexer che seleziona l'indirizzo che verrà scritto nel Program Counter. Viene impostato a 1 nel caso di un'istruzione di jump, 0 altrimenti.

Branch: Insieme al segnale di jump, comanda il multiplexer che seleziona l'indirizzo che verrà scritto nel PC. Prende il valore di 1 nel caso di un'istruzione beq e del segnale di zero dell'ALU alto o nel caso di un'istruzione bne e del segnale di zero dell'ALU basso. In questo caso l'indirizzo di salto viene calcolato aggiungendo il valore immediato dell'istruzione al valore del PC+4.

MemRead: Nell'istruzione di lw è affermato, comanda la memoria centrale a lavorare in fase di lettura: dato l'indirizzo in input, viene caricato il contenuto della relativa word in output della memoria.

Mem2Reg: Comanda il multiplexer che seleziona il dato da scrivere nel registro rd (nel caso di istruzioni di tipo R) o rt (nel caso di istruzioni di tipo I). Con segnale alto,

nel caso dell'istruzione di lw, il dato da scrivere viene caricato da memoria. Con segnale basso viene prelevato dall'output dell'ALU.
NB: il dato viene scritto nel registro specificato nell'istruzione solo con il segnale di RegWrite alto.

ALUs: Specifica l'operazione che andrà a svolgere l'ALU o specifica che l'operazione deve essere determinata dai bit del campo funct dell'istruzione. E' un segnale di controllo a 3 bit, permette quindi di combinare fino a 7 operazioni diverse di tipo immediate. Una combinazione è riservata alle operazioni di tipo R.

MemWrite: Quando il suo valore è alto permette la scrittura in memoria centrale. Il contenuto in ingresso al Memory Data Register viene memorizzato nella cella il cui indirizzo è caricato nel Memory Address Register.

ALUSrc: Quando ha valore basso il secondo operando dell'ALU proviene dalla seconda uscita in lettura del Register File(DataRead2). Quando ha valore alto il secondo operando dell'ALU è la versione estesa (con segno) del campo immediate.

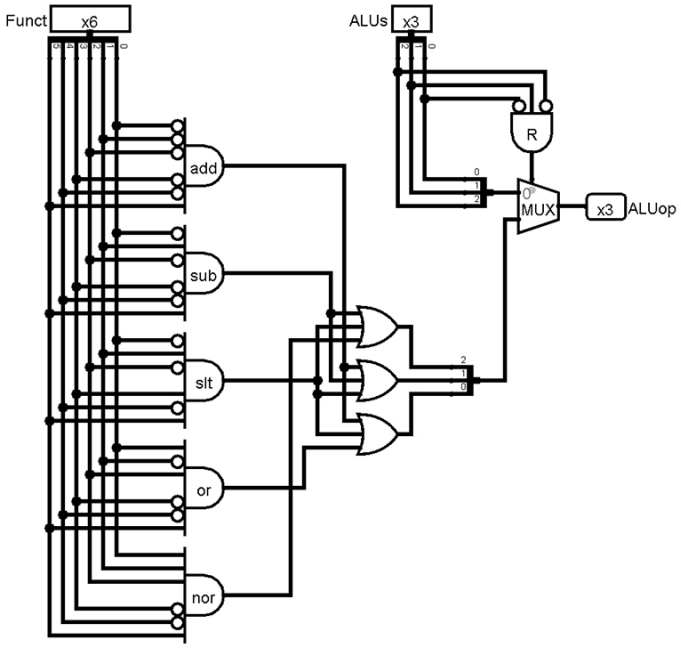
RegWrite: Quando il suo valore è alto permette la scrittura sui registri del Register File: il registro indicato all'ingresso RegWrite prende il valore dell'input presente in DataWr.

5 Registri

PC(Program Counter): contiene l'indirizzo dell'istruzione successiva da prelevare dall'IR.
$0-$31: 32 registri "general purpose" modificabili dalle istruzioni in esecuzione.

6 Unità di controllo dell'ALU

L'unità di controllo principale decodifica i bit del campo op dell'istruzione. Quando questi bit non sono tutti 0, il segnale di ALUs ricevuto dall'unità di controllo specifica l'operazione dell'ALU e il segnale viene passato all'ALU senza modifiche(il valore di ALUs sarà uguale a quello di ALUOp). Altrimenti quando l'opCode è uguale a 0b000000 l'istruzione in questione è di tipo R. In questo caso l'unità di controllo dell'ALU ricaverà, in funzione dei bit del campo funct, l'operazione che l'ALU andrà ad eseguire. Il segnale di ALUs è quindi un codice speciale che indica all'unità di controllo dell'ALU se l'operazione è da determinare dai bit del campo funct.



	funct	ALUs	ALUop
add	100000	010	011
addi	X	011	011
sw	X	011	011
lw	X	011	011
sub	100010	010	110
beq	X	110	110
bne	X	110	110
slt	101010	010	111
slti	X	111	111
or	100101	010	001
ori	X	001	001
and	100100	010	000
andi	X	0	000
nor	100111	010	100
j	X	X	X

Figura 8: Circuito e tabella di verità ALU_UC

7 ALU

L'Unità Aritmetico Logica ha la funzione di calcolo tra due input a 32bit che possono provenire da due registri o da un registro e un immediato con segno esteso(Vedi paragrafo 3.1 e 4.1). La finalità del calcolo dell'ALU non è sempre legata al significato intrinseco dell'istruzione, infatti oltre ad eseguire l'operazione aritmetica o logica che può essere specificata da istruzioni come add, sub, and, or (e le relative istruzioni immediate), l'ALU può:

- Eseguire una sottrazione per confrontare due numeri nel caso d'istruzioni di branch(beq, bne). In questa circostanza diventa funzionale il segnale di zero in uscita dall'ALU che insieme al segnale di controllo di Branch determina se verrà effettuato o meno il salto condizionato.
- Calcolare un indirizzo di memoria aggiungendo il valore di un registro al campo immediato esteso di segno nel caso di istruzioni di memoria(lw,sw).

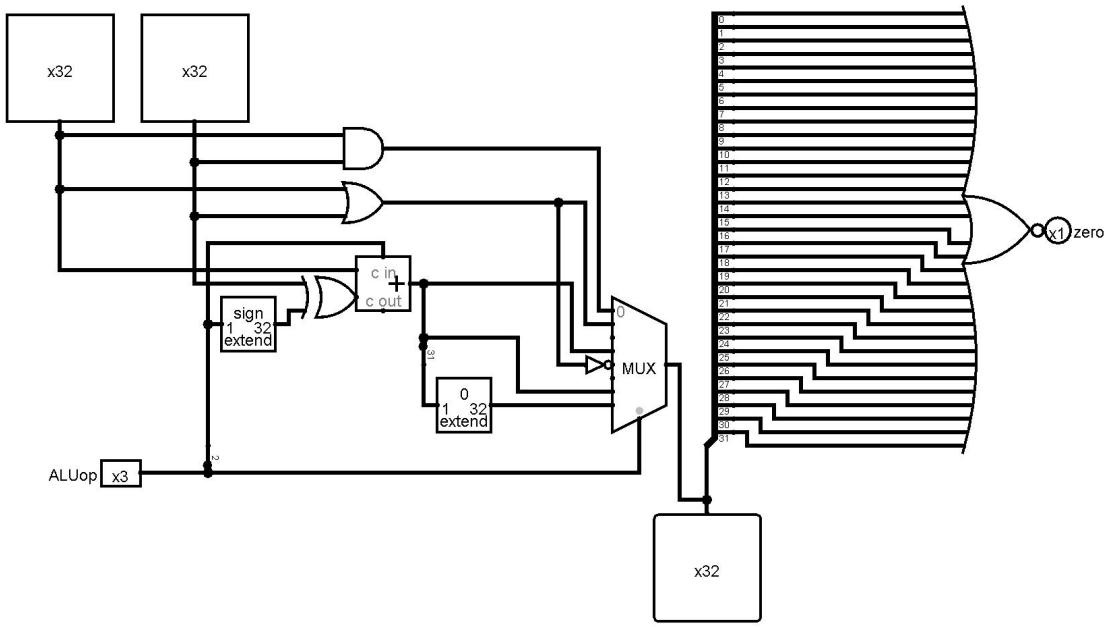


Figura 9: Circuito ALU

Nota implementativa: la sottrazione è implementata con lo stesso full adder usato per l'operazione di somma. Lo xor che porta i bit alla seconda entrata del full adder, complementa i bit del secondo operando solo nel caso in cui l'ALUop indica una sottrazione, mentre li lascia inalterati nel caso della somma. Al riporto in ingresso del full adder entra il bit più significativo dell'ALUop: nel caso della somma sarà impostato a 0, altrimenti prenderà il valore 1(Vedi tabella di verità figura 8). Le altre operazioni non vengono considerate data la funzione del multiplexer che prende in input il risultato di tutte le operazioni calcolate e va a selezionare in output solo quella desiderata.

8 Conclusioni e possibili implementazioni

Per rendere più funzionale e completo il progetto si possono apportare alcune modifiche aggiungendo istruzioni all'ISA e, a discapito della complessità del circuito, renderlo più performante introducendo tecniche di pipelining e caching. Si può inoltre introdurre un traduttore hardware d'istruzioni assembly, un coprocessore per le operazioni floating point e uno per la gestione delle eccezioni(es. overflow, interrupts, illegal address ecc.)