PROGETTO ARCHITETTURA DEI CALCOLATORI

Introduzione

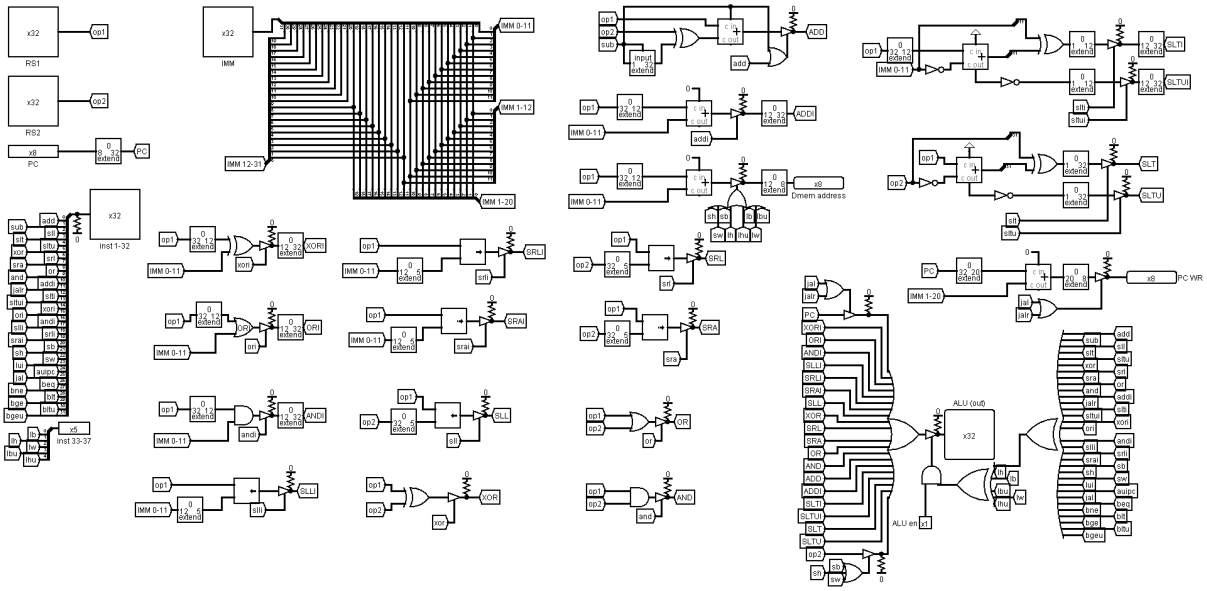
Questa tesina è stata scritta da:

Elena Maria Ciuffreda, matr. 119325
per il progetto di Architettura dei Calcolatori.

Obiettivo del progetto era quello di avere una CPU completa in-order implementata su Logisim, in grado di decodificare ed eseguire istruzioni risc-v, con opportuna logica di controllo per attivare in modo corretto le varie componenti.

La cpu dovrà essere in grado di interagire con una memoria sia per la parte di fetch delle istruzioni che per il caricamento di dati sui registri con le istruzioni di load

ALU



L'immagine mostra l'intera struttura del percorso dati e della logica combinatoria che costituisce l'unità aritmetico-logica (ALU) e il sistema di generazione degli immediati, delle operazioni e dei segnali di scrittura utilizzati in un processore RISC-V a 32 bit realizzato in Logisim.

Lo schema è suddiviso in sezioni funzionali chiaramente riconoscibili: il blocco dei registri sorgente (RS1 e RS2), il PC con il relativo estensore di segno, l'unità di generazione degli immediati (con i vari formati I, S, B, U, J), un insieme di moduli dedicati alla realizzazione delle singole micro-operazioni (add, addi, sub, xor, xori, or, ori, and, andi, shift logico e aritmetico, confronti SLT/SLTU, operazioni di load/store), e infine il grande multiplexer centrale che seleziona l'uscita finale dell'ALU, insieme ai segnali di controllo collegati al decodificatore di istruzione.

1. Registri sorgente, PC e ingressi della ALU

In alto a sinistra troviamo i blocchi che rappresentano le due sorgenti dell'ALU:

- **RS1 (base)**: un registro a 32 bit, etichettato *x32*, che rappresenta il valore letto dal registro sorgente 1.
- **RS2**: un altro valore a 32 bit proveniente dal registro sorgente 2.
- **PC**: il Program Counter, rappresentato come un valore a 8 bit esteso a 32 tramite un estensore. Questo PC può essere utilizzato per calcolare indirizzi relativi nelle istruzioni come AUIPC, JAL, JALR e nei salti condizionati.

Ogni valore è portato in diversi sottocircuiti per eseguire operazioni differenti.

2. Generazione degli immediati

Al centro dell'immagine si osserva un grande blocco di combinazioni di bit che ricrea la logica di estrazione degli immediati dai diversi formati delle istruzioni RISC-V:

- **IMM 0–11**, **IMM 1–12**, **IMM 1–20**, ecc.: linee di bit assemblate a partire dal campo imm contenuto nell'istruzione.
- Lo schema mostra chiaramente la ricomposizione bit-per-bit tipica degli immediati RISC-V, soprattutto per i formati più complessi come **B-type** e **J-type**, in cui i bit non sono contigui nell'istruzione.

Da notare:

- Per il formato I (utilizzato da ADDI, ORI, ANDI, SLTI...), l'immediato è semplicemente i bit INST[31:20].
- Per il formato S (store), l'immediato è ricostruito concatenando INST[31:25] e INST[11:7].
- Per il formato B (branch), l'immediato è composto da bit sparsi: INST[31], INST[7], INST[30:25], INST[11:8], più uno zero finale.
- Analogamente il formato J è ricomposto riordinando opportunamente i bit.

Tutti questi immediati vengono poi estesi con segno (o zero) tramite i blocchi "0->32 extend" o "1->2->32 extend", a seconda del tipo di estensione richiesta.

3. Moduli elementari delle singole operazioni

La parte centrale e inferiore dell'immagine è popolata da una lunga serie di sottocircuiti, ognuno dei quali implementa una *specifica operazione* RISC-V.
Questi moduli sono completamente separati e ricevono in ingresso:

- *op1* → tipicamente RS1 o un immediato
- *op2* → RS2 o un immediato
- l'estensione dell'immediato (quando necessaria)
- eventuali segnali di carry out (per addizioni e sottrazioni)

3.1 Operazioni aritmetiche

Sono presenti sottocircuiti per:

- **ADD**: addizione operando RS1 + RS2.
  Utilizza un full adder a 32 bit e genera anche un carry out.
- **ADDI**: addizione con immediato.Riceve op1 e un immediato esteso.

- **SUB**: implementata tramite ADD con inversione del secondo operando e carry-in impostato a 1.
- ***AUIPC****: non mostrata direttamente, ma parte del calcolo PC + imm.*

3.2 Operazioni logiche

Sono implementate singolarmente:

- **XOR / XORI**
- **OR / ORI**
- **AND / ANDI**

Ognuna usa il proprio gate logico e un estensore per l'immediato, quando necessario.

3.3 Operazioni di shift

Sono presenti ben sei moduli distinti:

- **SLL**: shift logico a sinistra, con RS2 come contatore.
- **SLLI**: shift logico a sinistra con immediato.
- **SRL / SRLI**: shift logico a destra.
- **SRA / SRAI**: shift aritmetico a destra, preservando il segno.

Ogni blocco contiene uno shifter e un'estensione corretta dei bit shiftati.

3.4 Operazioni di confronto (Set-Less-Than)

Due moduli fondamentali:

- **SLT**: confronto signed
- **SLTU**: confronto unsigned

Questi sottocircuiti analizzano il risultato di RS1 − RS2 tramite carry out o segnali combinatori, e producono un risultato booleano di 32 bit (0…0 o 0…1).

3.5 Operazioni di load/store

Sono presenti i moduli per costruire gli indirizzi di memoria di LOAD/STORE:

- Modulo che calcola:
  DMem Address = RS1 + immediato
- Moduli che generano i segnali di store:
  sw, sh, sb, lui, lhu, lw con estensioni di segno o zero laddove richiesto.

3.6 Operazioni di salto

Sono presenti i moduli:

- **JAL**: PC + immediato
- **JALR**: (RS1 + immediato) & ~1

Entrambe producono un nuovo PC e un valore da scrivere nel registro di ritorno.

4. Grande Multiplexer di Selezione dell'ALU

A destra troviamo il componente più importante dell'intero schema: un grande multiplexer che raccoglie tutte le uscite dei sottocircuiti visti in precedenza.

Ciascun filo in ingresso è etichettato:

- ADD
- ADDI
- SUB
- XOR
- XORI
- AND
- ANDI
- OR
- ORI
- SLL
- SLLI
- SRL
- SRLI
- SRA
- SRAI
- SLT
- SLTI
- SLTU
- SLTUI
- JAL
- JALR
- LB, LBU, LH, LHU, LW
- SB, SH, SW
- BEQ, BNE, BLT, BGE, BLTU, BGEU

Questa lunga lista rappresenta tutte le operazioni implementabili dalla ALU.
Il multiplexer riceve anche un segnale di abilitazione **ALU en x1** e produce come uscita:

- **ALU(out)**: un valore a 32 bit
- segnali aggiuntivi verso i moduli di load/store e verso il PC update.

A valle del multiplexer, un ulteriore stadio seleziona eventuali scritture PC → PC WR.

5. Decoder delle istruzioni

A sinistra nella parte medio-bassa c'è un blocco "inst-1-32" che elenca tutte le istruzioni decodificate:

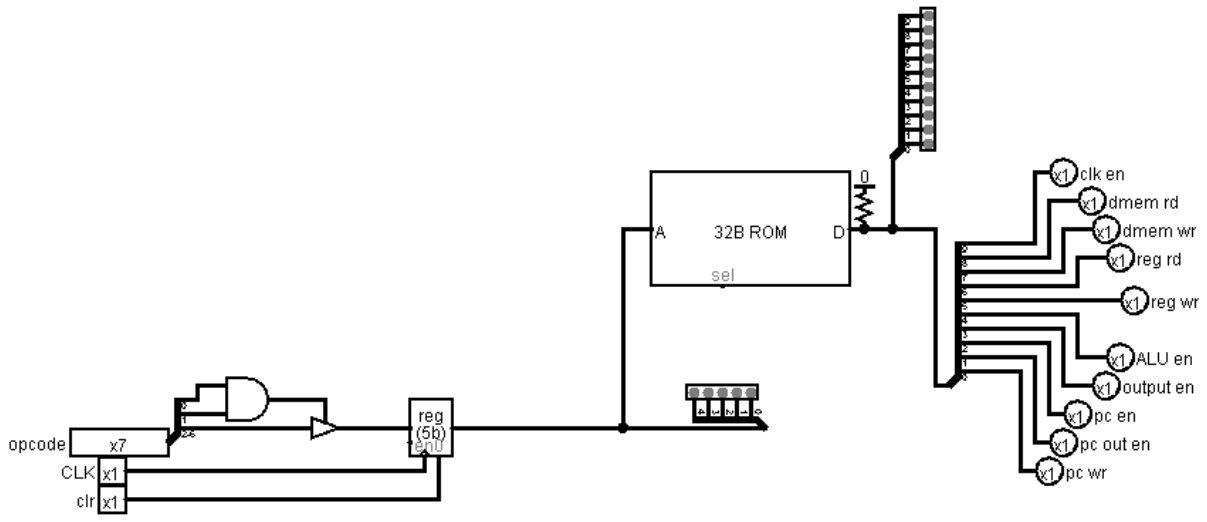
- add, sub, slt, sltu
- xor, srl, sra
- addi, ori, andi
- load (lb, lbu, lh, lhu, lw)
- store (sb, sh, sw)
- branch (beq, bne, blt, bge, bltu, bgeu)
- auipc, jal, jalr, lui

Questo blocco funge da *control unit*: da esso escono i segnali che andranno a pilotare il grande multiplexer e gli altri selettori.

Per quanto questo componente sia stato pensato in modo '*didattico*', mi ha permesso di vedere ogni operazione isolata in un blocco autonomo - senza dover passare per una singola ALU multifunzione.

Control Logic

Implementazione di una logica di controllo minimale



Sezione di ingresso: opcode, clock e clear

La parte sinistra dello schema mostra i tre segnali di ingresso principali:

- **opcode (x7)**: è un bus composto da 7 bit. Questo numero di bit è sufficiente per identificare fino a 128 possibili istruzioni.Il bus degli opcode rappresenta la parte alta dell'istruzione, cioè quella che, tradizionalmente, definisce la classe dell'operazione da eseguire (aritmetica, accesso a memoria, salto, ecc.).
- **CLK (x1)**: è il segnale di clock, necessario per sincronizzare il comportamento del registro interno che appare più avanti nel diagramma.
- **clr (x1)**: è un segnale di reset asincrono o sincrono utilizzato per riportare lo stato interno della macchina in una condizione iniziale nota.

Questi tre segnali alimentano direttamente il blocco centrale della logica di controllo.

2. Blocco combinatorio locale: AND + buffer invertente

Subito dopo l'ingresso dell'opcode compare una piccola logica combinatoria composta da una porta **AND** e da un **buffer invertente**. Questa sezione svolge un ruolo essenziale: determina se il valore dell'opcode debba effettivamente essere registrato nel ciclo corrente.

È visibile una porta AND a due ingressi, uno dei quali preleva un bit specifico dell'opcode, mentre l'altro ingresso potrebbe essere collegato a un segnale di abilitazione interno.

La funzione della porta è probabilmente quella di selezionare un sottoinsieme di istruzioni che richiedono un aggiornamento del registro successivo.

L'invertitore capovolge questo segnale, producendo un impulso logico che abiliterà o disabiliterà il latch o il registro che appaiono a valle (invece di memorizzare sempre l'opcode, lo si registra solo quando serve, evitando stati inconsistenti e garantendo che il bus che entra nel decoder sia stabile e pulito).

3. Registro a 5 bit (reg 5b)

Proseguendo verso destra si incontra un **registro da 5 bit** ("reg 5b"). Il numero di bit suggerisce che non viene memorizzato l'intero opcode da 7 bit, ma solo una sua parte significativa.

Il controllo interno si basa su un sottoinsieme dell'opcode, magari perché le istruzioni sono raggruppate per classi e la decodifica fine è demandata ad altri moduli.

Il registro riceve:

- in ingresso dati i bit selezionati dell'opcode,
- in ingresso di controllo il clock,
- in ingresso di clear il segnale di reset,
- in ingresso di enable il segnale proveniente dalla combinazione AND + invertitore.

In altre parole, questo registro trattiene la versione "pulita" dell'opcode che servirà per la decodifica tramite ROM (come detto precedentemente) .

4. Bus intermedio e sonda

L'uscita del registro confluisce su un punto di raccordo dotato di una **sonda visiva** (un hex-display). Questo elemento aiuta a mostrare in tempo reale il valore attualmente contenuto nel registro e permette allo studente di verificare quale opcode è stato effettivamente catturato dal sistema nell'ultimo ciclo.

5. ROM da 32 byte con decodifica dei segnali di controllo

Al centro del diagramma appare una **ROM da 32 byte**. Questa ROM svolge una funzione fondamentale: agisce come tabella di decodifica. Ogni possibile valore a 5 bit proveniente dal registro seleziona una riga della ROM, e la ROM restituisce in uscita un vettore di controllo composto da più bit.

La dicitura *"32B ROM"* indica che ci sono 32 indirizzi (compatibile con i 5 bit del registro) e ciascun indirizzo contiene una parola più larga di 8 bit—infatti la ROM deve generare molti segnali di controllo contemporaneamente.

L'uscita della ROM, indicata dal terminale **D**, viene connessa a un bus più largo che raggiunge il blocco dei segnali di controllo sulla destra.

Il piccolo elemento grafico sul terminale D, simile a una resistenza tirata a zero o a un generatore costante, indica la presenza di un **pull-down** o un default a zero. Ciò significa che, in assenza di una selezione valida, la ROM produce un vettore nullo, evitando attivazioni incontrollate dei segnali di controllo.

6. Multiplexer o selettore

Sotto la ROM è presente un piccolo multiplexer/selector. Questo può servire per una delle seguenti ragioni:

- permettere di scegliere una modalità di debug o test,
- selezionare tra più insiemi di segnali di controllo,
- scegliere se inviare output reali o simulati.

7. Uscita della ROM e bus dei segnali di controllo

La parte destra dello schema è dedicata all'esposizione dei signals generati dalla ROM. Il vettore di uscita viene scomposto visivamente in linee singole, ognuna delle quali raggiunge un LED o un'etichetta corrispondente al segnale di controllo specifico.

I segnali elencati sono:

- **clk en**: abilitazione del clock interno o di una parte del datapath.
- **dmem rd**: abilitazione della lettura dalla memoria dati.
- **dmem wr**: abilitazione della scrittura nella memoria dati.
- **reg rd**: segnale che comanda la lettura dal registro selezionato.

- **reg wr**: segnale che abilita la scrittura nel file di registri.

- **ALU en**: abilitazione dell'unità aritmetico-logica.
- **output en**: consente di inviare il risultato verso un eventuale bus di output.
- **pc en**: abilitazione generale del program counter.
- **pc out en**: consente al program counter di mettere il proprio valore sul bus.
- **pc wr**: abilita la scrittura nel program counter (tipicamente per salti e branch).

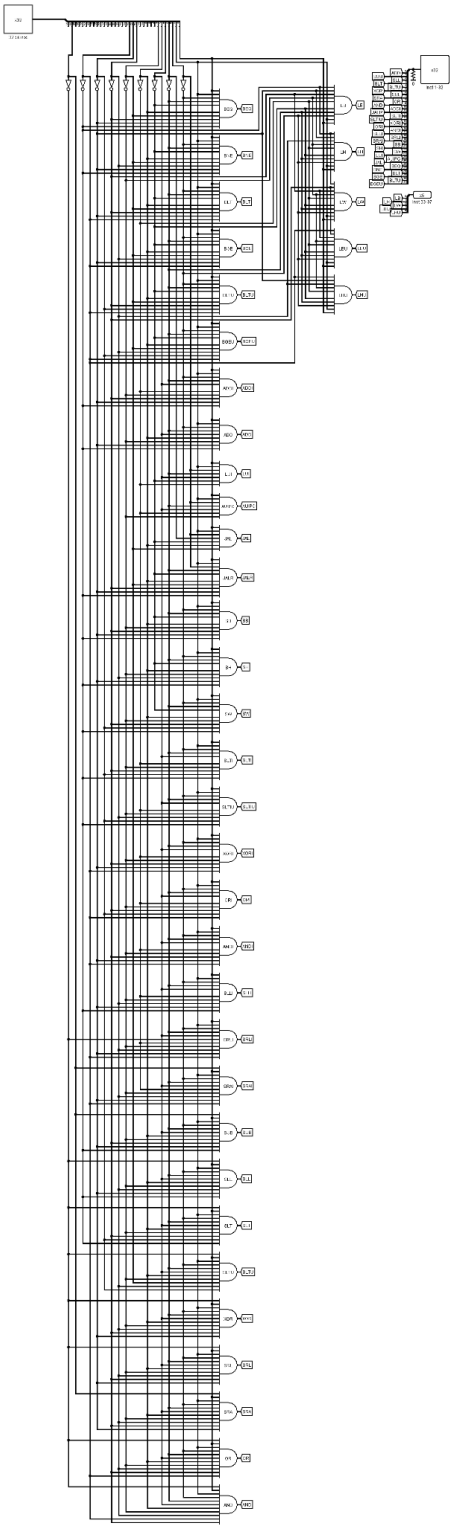
8. Struttura complessiva

L'intero circuito può essere interpretato come una pipeline a due fasi:

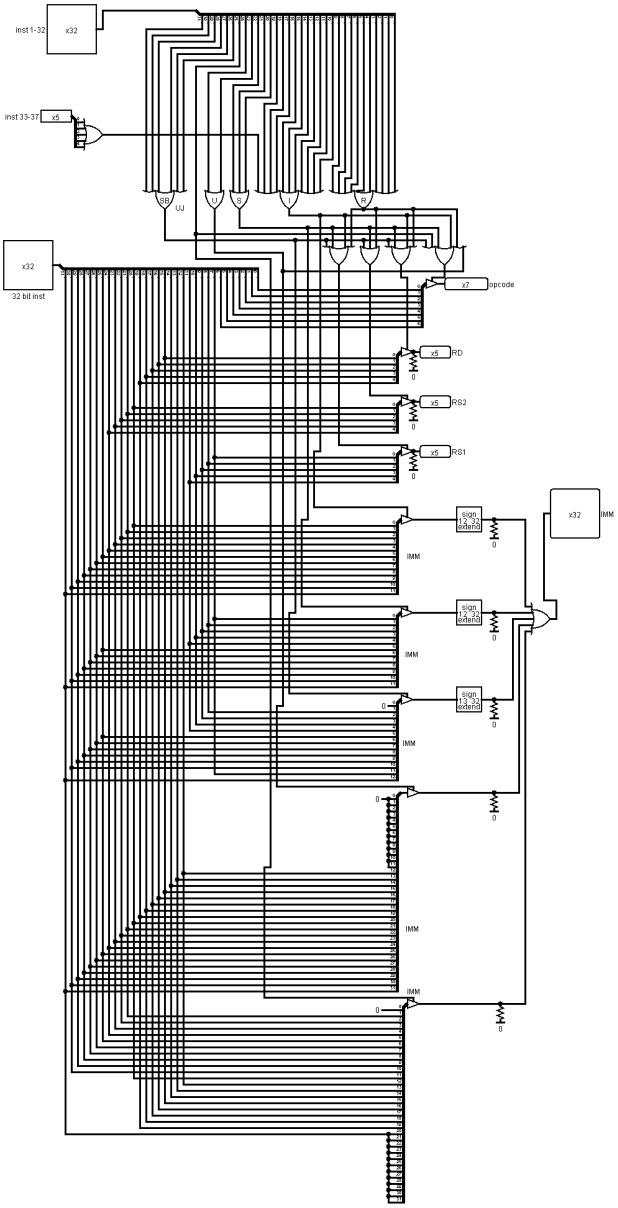
1. fase sincrona: cattura dell'opcode in un registro stabile;
2. fase combinatoria: decodifica dell'opcode tramite ROM per generare i segnali di controllo.

Il vantaggio di questa architettura è la sua chiarezza didattica:ogni operazione è ben separata.

Decoder



decoder 1



decoder 2

L'immagine a sinistra (*decoder 1*) rappresenta il **decoder delle istruzioni** costruito in stile *gate-level* .
Il circuito svolge la funzione centrale del datapath di controllo: a partire dai **32 bit dell'istruzione in ingresso** (forniti dal blocco "32-bit inst"), genera una serie molto estesa di **segnali di controllo**, ognuno dei quali attiva una specifica micro-operazione dell'unità computazionale o dell'unità di memoria.

Struttura generale del decoder 1

Nella parte superiore sinistra si trova l'ingresso a 32 bit dell'istruzione. Da questo bus si dipartono numerose linee, ciascuna collegata a elementi logici elementari (principalmente NOT, AND, OR) che analizzano i campi dell'istruzione secondo il formato RISC-V.

L'immagine mostra che il decoder non impiega un singolo blocco "ROM" o un controllo microprogrammato, ma una *rete completamente combinatoria,* dove ogni condizione sul campo opcode, funct3 e funct7 viene tradotta manualmente in una congiunzione di segnali logici.

L'insieme delle linee verticali centrali rappresenta il bus dei singoli bit dell'istruzione, ognuno inviato a più porte logiche che testano condizioni relative ai diversi campi di codifica. I bit più alti e quelli centrali sono duplicati o invertiti.

Decoder delle istruzioni e riconoscimento delle classi

Procedendo verso il basso, compaiono una serie molto numerosa di porte logiche etichettate con nomi standard delle istruzioni RISC-V, come ad esempio:

- **BEQ, BNE, BLT, BGE, BLTU, BGEU**, che appartengono alla classe delle istruzioni di *branch* (B-type).
- **LB, LH, LW, LBU, LHU**, che appartengono al gruppo dei *load* (I-type).
- **SB, SH, SW**, che riconducono alle istruzioni di *store* (S-type).
- **ADD, SUB, AND, OR, XOR, SLL, SRL, SRA**, appartenenti alle istruzioni aritmetico-logiche R-type.
- **ADDI, ORI, ANDI, SLTI**, e così via, facenti parte delle versioni immediate I-type.
- **LUI** e **AUIPC**, che appartengono alle istruzioni U-type.
- **JAL** e **JALR**, appartenenti alle istruzioni di salto (J-type e I-type).

Ogni istruzione è rappresentata da una **porta AND** che combina:

1. alcune linee provenienti dall'opcode

2. eventualmente il campo funct3,
3. in molti casi il funct7,
4. ed eventuali versioni invertite di questi segnali.

In questo modo, ogni porta AND “riconosce” l’unica combinazione precisa di bit che identifica una particolare istruzione.

Generazione dei segnali di controllo

L’uscita di ciascuna porta AND è un segnale di controllo che attiva:

- un percorso dati dell’ALU (es. ADD, SUB, SLL, SRL, AND...),
- una modalità di accesso alla memoria (es. LB, LH, LW),
- un tipo di branch,
- un salto PC-relative,
- oppure segnali specifici dell’unità di scrittura del registro (RegWrite),
- e segnali per la formazione dell’immediato.

L’immagine mostra che le uscite istruzione-per-istruzione vengono poi raggruppate in un bus orientato verso destra, dove confluiscono circuiti legati alla generazione degli immediati.

Per quanto concerne il *decoder2 (immagine a dx, a pagina 1)* rappresenta un blocco fondamentale: il modulo di estrazione dei campi dell’istruzione (opcode, registri, immediati) e la generazione dei diversi formati di immediato I, S, B, U e J.

1. Struttura generale del decoder 2

In alto a sinistra compare un ingresso etichettato “inst 1–32”, cioè il bus a 32 bit dell’istruzione completa.

Da questo bus si diramano numerose linee che raggiungono porte logiche e multiplexer a valle.

Subito sotto si nota un secondo gruppo di linee, qui etichettato come *“Inst 33–37”,* collegato a un gruppo di cinque bit (x5).

- R-type
- I-type
- S-type
- B-type
- U/J-type

Le porte OR e AND immediatamente successive aggregano i bit dell'opcode e li *filtrano* nei rispettivi formati (ad esempio S, B, UJ, ecc.).

Si tratta di una pre-decodifica che permette di attivare solo i percorsi necessari per la ricostruzione dell'immediato.

2. Estrazione di opcode e registri

A metà immagine si osserva il blocco più denso di cablaggi: da qui vengono estratti tre campi critici:

- RD (destinazione) – 5 bit
- RS1 – 5 bit
- RS2 – 5 bit

Ogni campo è indirizzato verso un multiplexer che restituisce i 5 bit assegnati, insieme ad alcune costanti (tipicamente 0).
(Il motivo per cui compaiono costanti 0 in parallelo ai registri sorgente è semplice: alcuni formati di istruzione (ad esempio I-type, U-type o JAL) non prevedono RS2 o RD in posizioni convenzionali, perciò il circuito deve poter fornire un valore nullo quando il campo non è definito.)

Analogamente, l'opcode è estratto attraverso un bus a 7 bit che scende dalla sezione superiore e viene riportato verso destra tramite un blocco x7.

3. Generazione degli immediati

La parte più complessa e interessante dell'immagine è la sezione inferiore, dove si generano i diversi tipi di immediato, ovvero i valori numerici estratti da specifici campi dell'istruzione.

Lo schema mostra chiaramente:

- Immediate I-type
- Immediate S-type
- Immediate B-type
- Immediate U-type
- Immediate J-type

Ognuno di questi immediati ha uno schema di bit completamente diverso, e lo schema illustrato ricostruisce fedelmente ciascuno, bit per bit, collegando manualmente i campi inst[31], inst[30:25], inst[24:20], inst[19:12], ecc.

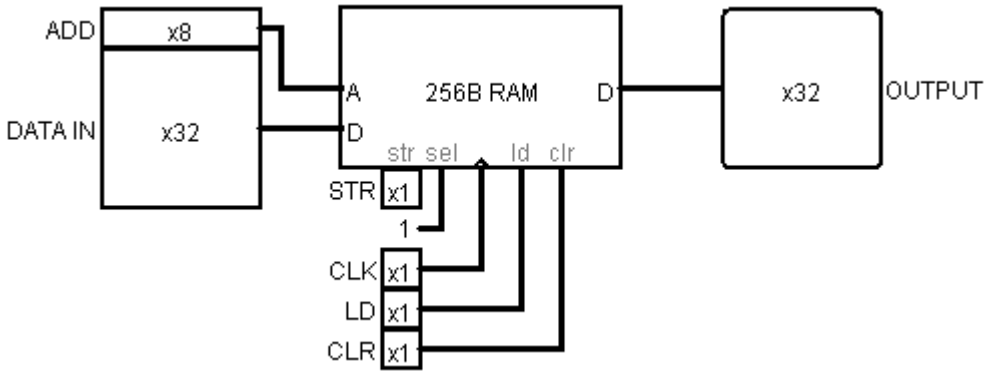
È molto evidente che:

- ogni immediato parziale (IMM) viene elaborato in uscita tramite un sign extender,
- i diversi immediati vengono inviati a un multiplexer finale,
- il multiplexer sceglie quale formato restituire in base al tipo di istruzione precedentemente riconosciuto.

Tutti gli immediati devono essere estesi a 32 bit prima di essere utilizzati dalla ALU.

Per completezza ho aggiunto nella cartella un file (Supported Instructions.txt) nel quale è riportato l'elenco completo delle istruzioni RISC-V effettivamente riconosciute dal mio decoder e gestite dal datapath del processore.

DMEM



L’immagine rappresenta uno schema funzionale di un sottosistema di memoria.

Lo schema mostra l’interazione fra tre elementi principali: un modulo di input che fornisce indirizzo e dato da scrivere, un blocco centrale che incapsula una memoria RAM da 256 byte, e un modulo di uscita che riceve il dato letto dalla memoria.

L’obiettivo del diagramma è illustrare come avviene il flusso dell’informazione da e verso la memoria tramite i segnali di controllo.

Struttura generale del componente DMEM

A sinistra compare un blocco che aggrega due segnali:

uno è etichettato “ADD” ed è largo 8 bit (x8),l’altro è etichettato “DATA IN” ed è largo 32 bit (x32).

L'indirizzo di 8 bit consente di selezionare una delle 256 celle della RAM, mentre il valore a 32 bit rappresenta il dato da memorizzare.

Questo è coerente con la dicitura “256B RAM”, che indica infatti una RAM organizzata in 256 parole da 32 bit ciascuna.

Dal blocco dell’input partono due linee dirette verso la RAM: una porta l’indirizzo (A), l’altra porta il dato da scrivere (D). Al centro dell’immagine, infatti, compare il blocco principale: la “256B RAM”, che è dotata di una porta A per l’indirizzo, una porta D per l’ingresso dati e una porta D anche per l’uscita dati.

Oltre a queste porte principali, la RAM presenta una serie di segnali di controllo fondamentali: STR (strobe), CLK (clock), LD (load) e CLR (clear). Questi segnali sono rappresentati come x1, ossia larghezza unitaria, perché si tratta di linee di controllo booleano che governano il comportamento interno della memoria.

Il segnale STR (strobe) è impiegato per abilitare l'accesso alla memoria: quando attivo, indica che l'operazione richiesta (lettura o scrittura) deve essere effettivamente eseguita. Il segnale CLK è il clock che sincronizza le operazioni di scrittura, tipico delle RAM sincrone: la scrittura avviene solo sul fronte desiderato del clock.

Il segnale LD (load) abilita la scrittura sulla memoria: se LD è attivo al momento giusto, il dato presente in ingresso viene memorizzato nella cella indicata dall'indirizzo. Infine, il segnale CLR permette di cancellare la memoria, o almeno di portare a zero il contenuto interno, a seconda dell'implementazione.

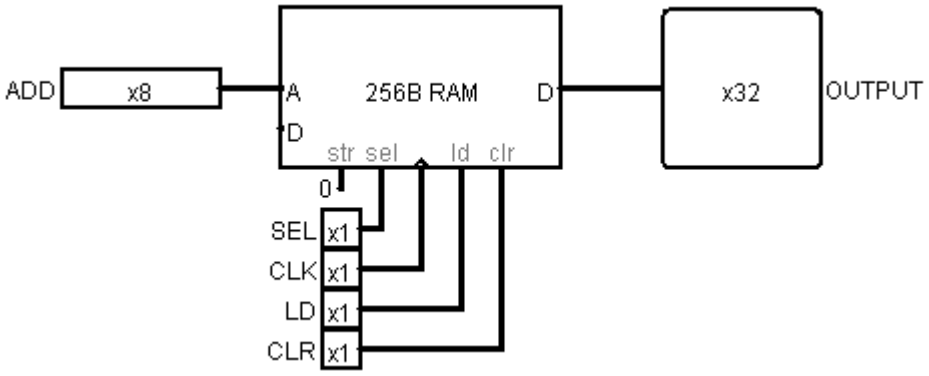
Sul lato destro del blocco RAM è presente l'uscita dei dati letti. L'uscita attraversa un ulteriore blocco etichettato "x32", a indicare che il dato in uscita è una parola a 32 bit. Da lì il segnale raggiunge il modulo **OUTPUT**, che fornisce il dato al resto del sistema.

Il file Python *1-255.py* presente all'interno della cartella è uno script utilizzato per **automatizzare la generazione dei valori esadecimali da 0 a 255**, formattati secondo lo standard atteso da un simulatore di memoria.

Lo script:

- apre un file chiamato *1to255.txt*
- scrive l'intestazione `"v2.0 raw\n"` necessaria per i file di memoria
- genera tutti i valori esadecimali in ordine crescente
- li esporta nel file in forma compatibile con una DMem/I-Mem
- termina una volta raggiunto il valore 0xFE (255−1)

IMEM



L'immagine rappresenta una micro–architettura che integra una **memoria RAM da 256 byte** con un piccolo insieme di segnali di controllo e due unità esterne dedicate alla preparazione dell'indirizzo e alla trasformazione del dato in uscita.

Struttura Generale

A sinistra è presente un blocco denominato **ADD**, che produce un valore di ingresso. Questo valore non entra direttamente nella RAM, ma attraversa un modulo etichettato **x8**.
Si tratta di un'**estensione a 8 bit**, cioè il valore prodotto da ADD viene portato a una lunghezza di 8 bit (o reinterpretato come un indirizzo a 8 bit), in modo da essere utilizzato come indirizzo per una RAM di dimensione **256 byte**, che richiede appunto **8 bit di indirizzo** ($2^8 = 256$).
La RAM ha:

- **Porta A (Address)**: collegata all'uscita del blocco x8, dunque riceve un indirizzo di 8 bit.
- **Porta D (Data)** in uscita: è il dato letto dalla RAM in corrispondenza dell'indirizzo indicato.
- **Porta D (Data input)**: dati in ingresso per eventuali operazioni di scrittura.
- **Linee di controllo interne**: nella parte bassa della RAM sono disegnati ingressi denominati:
    - **str**,
    - **sel**,
    - **A** (selezione indirizzo interno),
    - **ld** (load),
    - **clr** (clear).

Nella parte inferiore sinistra del diagramma compaiono quattro ingressi esterni:

Ognuno di questi ingressi passa attraverso un piccolo modulo etichettato **x1**, cioè un buffer o ripetitore a 1 bit.
Questo significa che :

- **CLK** governa il sincronismo delle operazioni di scrittura.
- **SEL** agisce come segnale che abilita l'accesso o seleziona il funzionamento della memoria.
  LD abilita il caricamento di un dato in ingresso verso una posizione della memoria.
- **CLR** consente l'azzeramento di celle o registri interni, utile nelle fasi di reset.

L'uscita dati **D** della RAM entra in un blocco etichettato **x32**. Il suffisso tipicamente indica un'estensione o una codifica a **32 bit**.
In pratica, l'informazione letta dalla memoria (probabilmente a 8 bit) viene:

- convertita,
- adattata,
- scalata o replicata

per ottenere un bus dati a **32 bit**, che costituisce l'output finale del sistema.

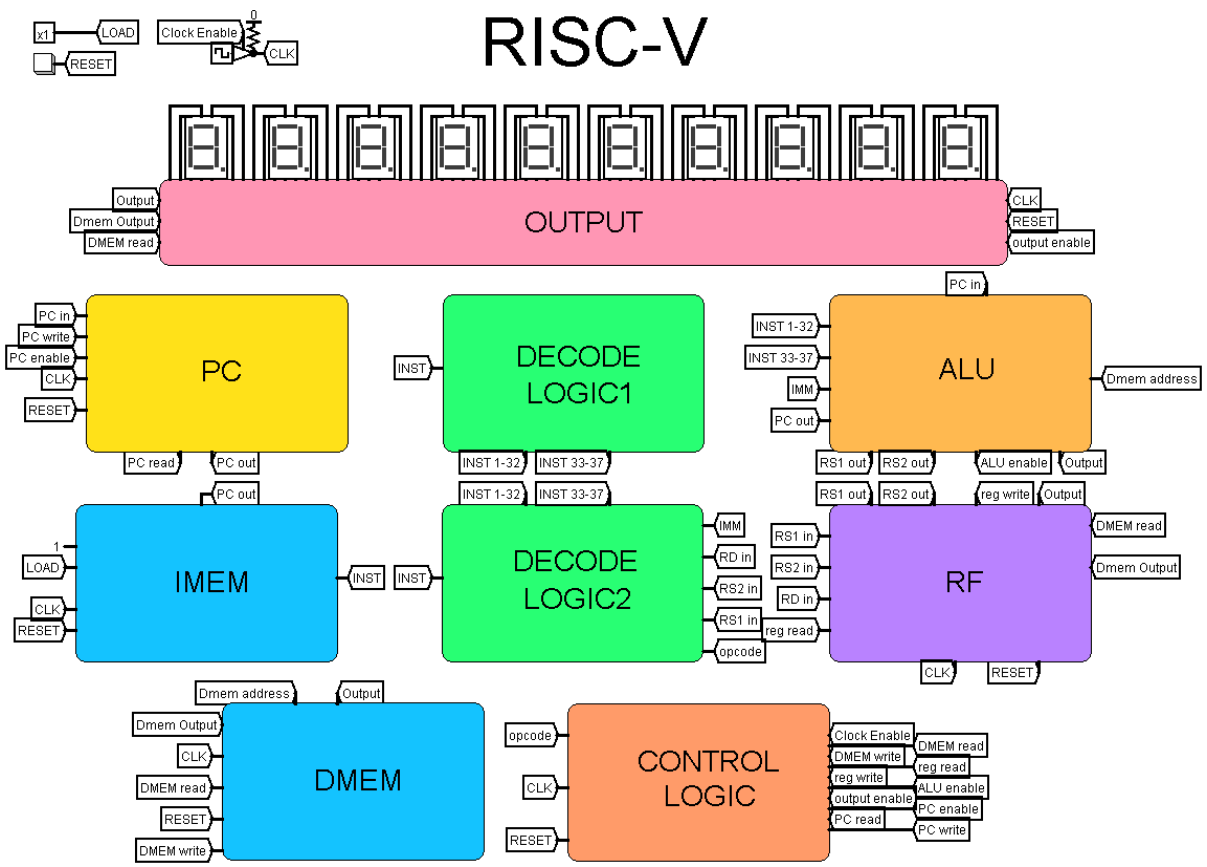
Il file *ADDI.txt* nella cartella raccoglie un insieme di istruzioni semplici, utilizzate come test preliminare per verificare il corretto caricamento e la decodifica della IMem in Logisim.

Il file bin to hex.py Si tratta dello script Python che hai creato per convertire il codice macchina generato da RARS (in binario) nel formato esadecimale richiesto dalla direttiva v2.0 raw di Logisim.

il file "Fibo" contiene il programma vero e proprio scelto come dimostrazione finale per il progetto. Ed è infatti il file che verrà inserito all'interno della ram. Contiene le istruzioni, già codificate, per l'algoritmo della serie di Fibonacci.

il file 'load' contiene un ulteriore test minimale, spesso usato nella fase di sviluppo della CPU per verificare la corretta esecuzione delle istruzioni di load/store.

Main



L'immagine fornisce la panoramica completa dell'architettura di una cpu RISC-V multi-ciclo semplificato, mostrando i principali componenti funzionali e i flussi di segnale necessari per l'esecuzione di un'istruzione.

Qui troveremo tutti i blocchi fondamentali dell'ISA RISC-V che sono presenti: Program Counter, Instruction Memory, Data Memory, unità di decodifica, Register File, ALU e Control Logic. La disposizione grafica enfatizza il percorso dei dati e la propagazione dei segnali di controllo.

1. Program Counter (PC)

In basso a sinistra, troviamo il blocco PC, responsabile di mantenere l'indirizzo dell'istruzione corrente. Il PC include numerosi ingressi di controllo:

- **PC in** e **PC write**, che consentono di aggiornare il contenuto del contatore nei cicli in cui è necessario;
- **PC enable**, utile nelle architetture multi-ciclo per congelare il PC;
- **RESET**, che riporta il PC allo stato iniziale;

- **CLK**, che allinea il suo comportamento al clock globale del sistema.

Il PC produce due segnali fondamentali: **PC read** e **PC out**.

2. Instruction Memory (IMEM)

L'IMEM occupa la parte bassa sinistra dello schema. È una memoria a sola lettura nel contesto dell'architettura funzionante, ma include un ingresso **LOAD** che consente il caricamento del programma tramite Logisim.
Riceve il valore del PC e restituisce il valore **INST**, cioè l'istruzione a 32 bit che verrà eseguita.La memoria supporta CLOCK e RESET, garantendo sincronizzazione con il resto della CPU.

3. Decode Logic 1 e Decode Logic 2

Lo schema prevede due blocchi distinti di decodifica, che riflettono la suddivisione logica dell'istruzione nei vari campi:

- **Decode Logic 1** produce i primi segnali derivati dal formato dell'istruzione: i campi funzionali (ad esempio funct3, funct7), i campi immediati e alcune porzioni dei registri.

- **Decode Logic 2** completa la decodifica restituendo:

    - gli indici RD, RS1, RS2,
    - il valore immediato **IMM**,
    - l'opcode, che è instradato verso la Control Logic.

4. Register File (RF)

Il Register File, situato in basso a destra, riceve dagli stadi di decodifica gli indici RS1, RS2 e RD.Produce i valori letti dai registri RS1 e RS2, inviati direttamente all'ALU.Dispone del segnale **reg write**, generato dalla Control Logic, che abilita la scrittura nel registro RD in fase di completamento dell'istruzione.Il RF lavora in sincronia con CLK e supporta un segnale di RESET.

5. ALU

L'ALU compare nella parte destra centrale. Riceve:

- i due operandi (RS1 out e RS2 out),

- l'immediato,
- porzioni dell'istruzione decodificata,
- l'eventuale PC out per istruzioni come AUIPC.

Restituisce il risultato dell'operazione aritmetica o logica e produce anche un indirizzo per la Data Memory quando l'istruzione è un load o store.
È controllata da un segnale **ALU enable** proveniente dalla Control Logic.

6. Data Memory (DMEM)

Situata nella parte bassa destra dello schema, la DMEM riceve l'indirizzo generato dall'ALU. Dispone dei segnali:

- **DMEM read**,
- **DMEM write**,
- **CLK** e **RESET**.

Produce il dato caricato dalla memoria, destinato al Register File quando serve (ad esempio in un LW).

7. Control Logic

La Control Logic coordina l'intero processore. Riceve l'opcode e in base ad esso genera:

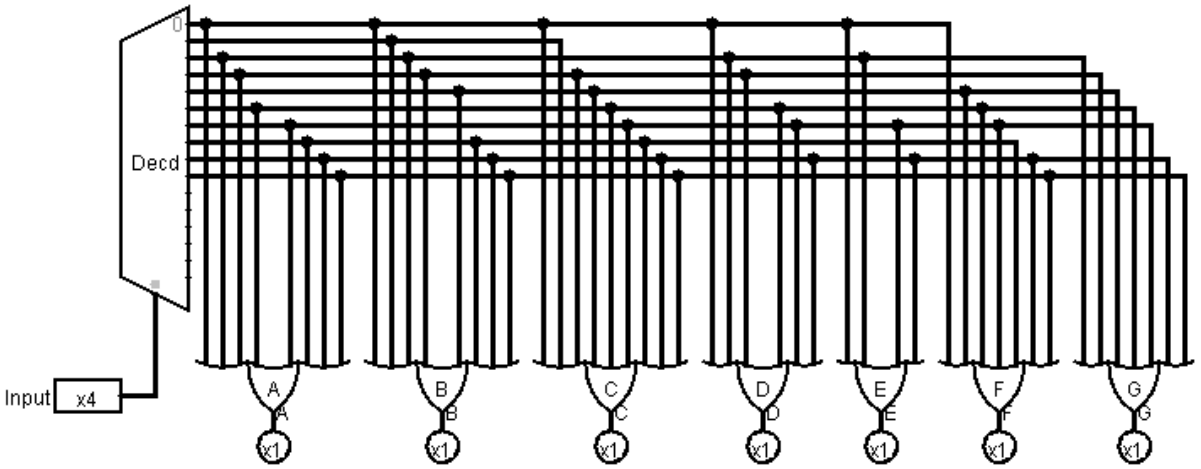
- reg write,
- DMEM read/write,
- PC write o PC enable,
- ALU enable,
- output enable.

È il componente che traduce l'ISA in segnali di controllo effettivi.

8. OUTPUT

La parte superiore dello schema contiene un grande blocco OUTPUT collegato sia all'ALU sia alla DMEM. Serve come uscita osservabile del processore per verificare i risultati delle istruzioni durante la simulazione.

Output DIsplay



L’immagine rappresenta un decoder per display a sette segmenti, progettato per convertire un ingresso binario a 4 bit nel corrispondente codice di attivazione dei segmenti A–G del display.

L’intero schema può essere interpretato come un esempio classico di realizzazione combinatoria tramite rete di AND-OR, costruita a partire da un decodificatore (o pre-decoder) posto sulla sinistra.

Il blocco etichettato “Decd” è un decodificatore 4->16: riceve in ingresso i quattro bit dell’input (generalmente BCD o una codifica binaria semplice) e genera sedici linee di uscita, ognuna attiva per una – e una sola – combinazione dei quattro bit.

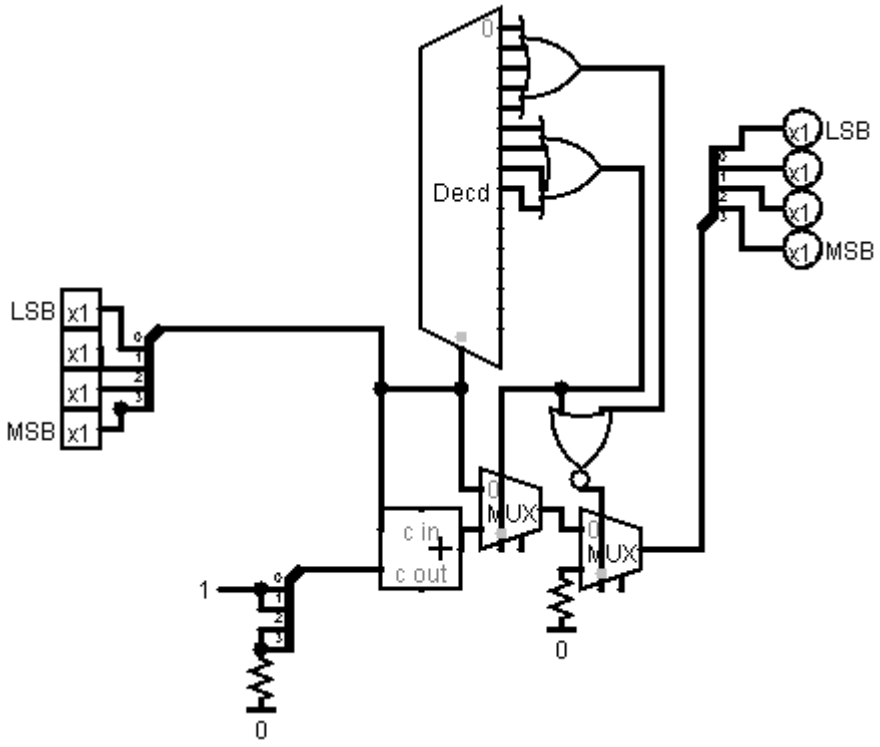
Queste sedici linee verticali, che dall’immagine si propagano verso destra, rappresentano le minterm della funzione Booleana: ogni colonna corrisponde a un valore possibile dell’ingresso, dal 0000 al 1111.

Per ogni segmento del display (A, B, C, D, E, F, G) è presente un insieme dedicato di porte OR, poste nella parte inferiore dello schema e identificate da un’etichetta sotto il simbolo logico.

Ogni OR raccoglie un certo numero di linee provenienti dal decodificatore: queste linee rappresentano tutte le combinazioni di input per cui quel particolare segmento deve accendersi. In pratica, ogni OR implementa la somma logica dei minterm relativi alla funzione che definisce l’accensione del segmento.

La parte più visibile dell’immagine è l’intreccio sistematico di collegamenti verticali e orizzontali: le uscite del decodificatore scorrono parallelamente, mentre linee orizzontali selezionate cadono verso le OR sottostanti.

Ogni porta OR finale ha un simbolo cerchiato “×1”, che indica che il segmento è controllato da un singolo segnale di uscita senza ulteriori pesi o moltiplicazioni: è semplicemente un’uscita combinatoria. L’insieme di queste sette uscite costituisce il pattern di accensione del display, coerente con le convenzioni standard di un 7-segmenti.



L’immagine rappresenta un circuito combinatorio che integra **decodifica**, **selezione tramite multiplexer** e **gestione di bit significativi** (MSB e LSB), con l’obiettivo di generare un’uscita corretta in base ai valori in ingresso e a un segnale di controllo esterno. Il diagramma mostra una struttura composta da tre elementi principali: la sezione di ingresso, il blocco di decodifica, e il percorso di uscita verso i due indicatori MSB e LSB.

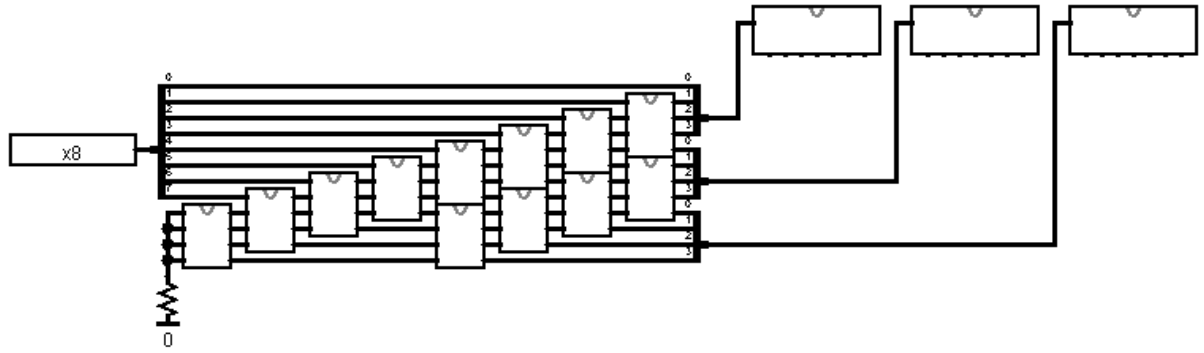
Nella parte sinistra sono visibili due ingressi etichettati come **LSB** e **MSB**, ciascuno replicato due volte tramite buffer a guadagno unitario (indicati con “×1”). Questi buffer rappresentano semplici ripetitori, utili per stabilizzare il segnale e consentire la sua distribuzione verso più nodi del circuito senza degradazioni. I due ingressi convergono in un nodo centrale che alimenta sia il decodificatore sia la logica sottostante.

Il blocco denominato **“Decd”** è un piccolo decodificatore combinatorio.

Le uscite del decodificatore vengono successivamente riconvogliate verso la parte destra del circuito, dove controllano i due segnali finali LSB e MSB.

Nella parte inferiore è presente un modulo contrassegnato con **c_in -> c_out**, che rappresenta un'unità di calcolo o di controllo di carry. Questo blocco riceve due ingressi: uno proveniente dai segnali LSB/MSB e uno fissato a 1 tramite resistenza (pull-up). Il percorso del segnale passa poi attraverso due **multiplexer**, ognuno configurato con un selettore binario derivato dalla logica del decodificatore. I multiplexer permettono di scegliere fra due possibili valori (tipicamente una costante logica 1 o 0), in base allo stato dell'ingresso.

La rete di collegamenti che ritorna verso destra integra i risultati del blocco di carry e dei multiplexer, stabilendo così quale valore debba essere inviato ai due indicatori finali. L'immagine mostra chiaramente che l'obiettivo del componente è determinare dinamicamente il valore dei bit più significativo e meno significativo.



L'immagine rappresenta una struttura logica che funge da **unità di selezione e instradamento dei segnali** verso più moduli di uscita. Lo schema è composto da tre sezioni principali: un insieme di linee d'ingresso parallele, una catena di blocchi logici intermedi, e un gruppo finale di moduli di visualizzazione o indicatori, posti sulla destra.

A sinistra compare un ingresso etichettato come **x8**, che indica la presenza di **otto linee di input parallele**. Queste linee si propagano orizzontalmente attraverso l'intero circuito, costituendo il bus principale. La parte inferiore mostra una serie di contatti collegati alla logica di terra (0).

Al centro si trova un insieme di moduli identici disposti in cascata **:** stiamo infatti parlando di un **latch**, perché ciascun modulo presenta più piedini d'ingresso e un'uscita controllata. l

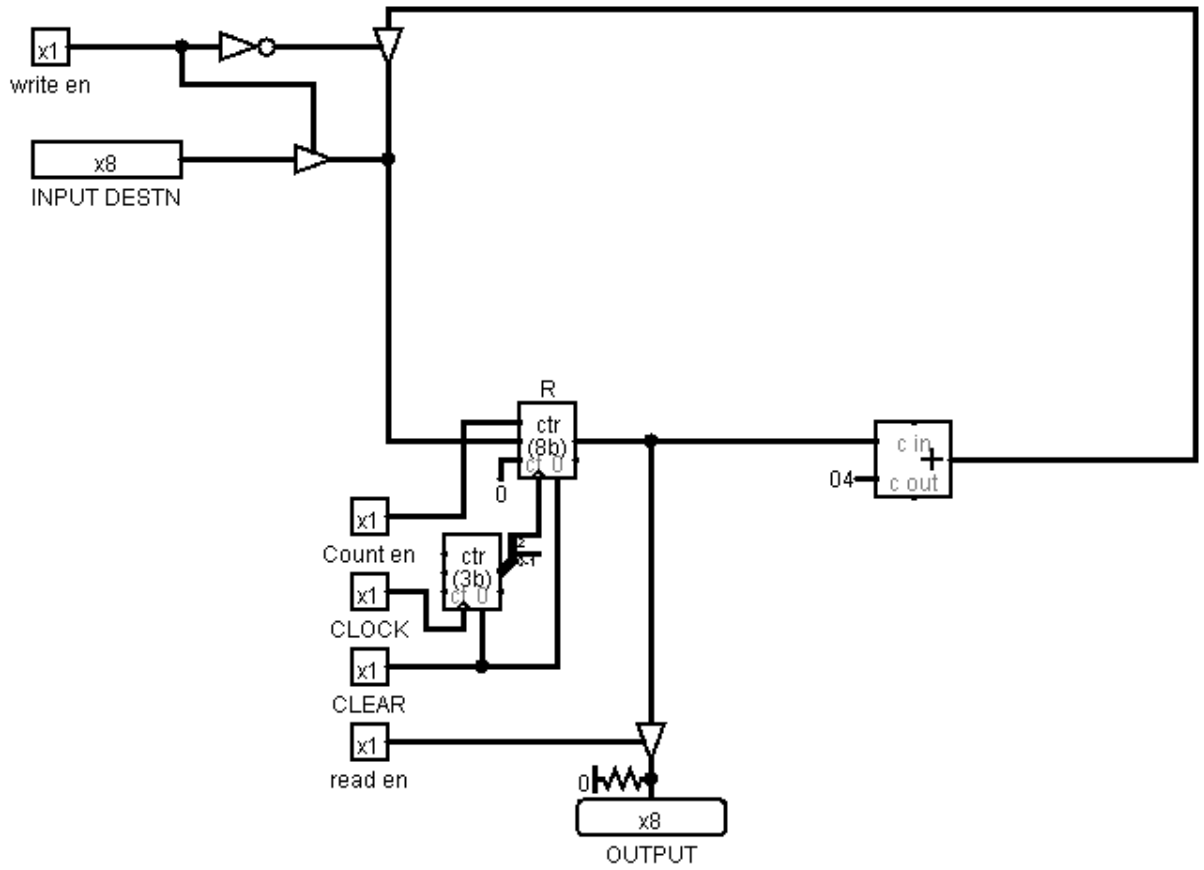
I blocchi sono connessi in modo tale da prelevare selettivamente alcuni bit del bus, trasformandoli in segnali più compatti che vengono poi raccolti sul lato destro del

circuito. La disposizione a "colla di pesce" delle linee verticali mostra che ogni modulo seleziona solo specifiche linee del bus a 8 bit.

Le otto linee, dopo essere passate attraverso la catena dei moduli interni, confluiscono in un blocco d'uscita che presenta tre connessioni distinte. Queste tre uscite alimentano i tre blocchi rettangolari posti sulla destra, che rappresentano verosimilmente **indicatori visivi o display**, ciascuno associato a un insieme di bit selezionati dal bus originale.

Nel complesso, l'immagine raffigura un circuito che **suddivide, filtra e inoltra porzioni di un bus a 8 bit verso diverse unità di visualizzazione**, utilizzando una sequenza ordinata di moduli logici intermedi per organizzare e indirizzare correttamente i segnali.

Program Counter



L’immagine rappresenta un circuito digitale che implementa una logica di scrittura, memorizzazione e lettura sequenziale di un valore a 8 bit, controllata da segnali di abilitazione e da un contatore interno.

A sinistra compaiono il segnale write enable e l’ingresso dati a 8 bit, denominato INPUT DESTN. Il write enable attraversa una piccola rete di porte logiche (inversione e AND) che determina se il dato deve essere inoltrato verso il nodo principale.

Il segnale viene combinato con la presenza effettiva dei bit in ingresso, producendo un’unica linea di controllo che governa il lato scrittura.

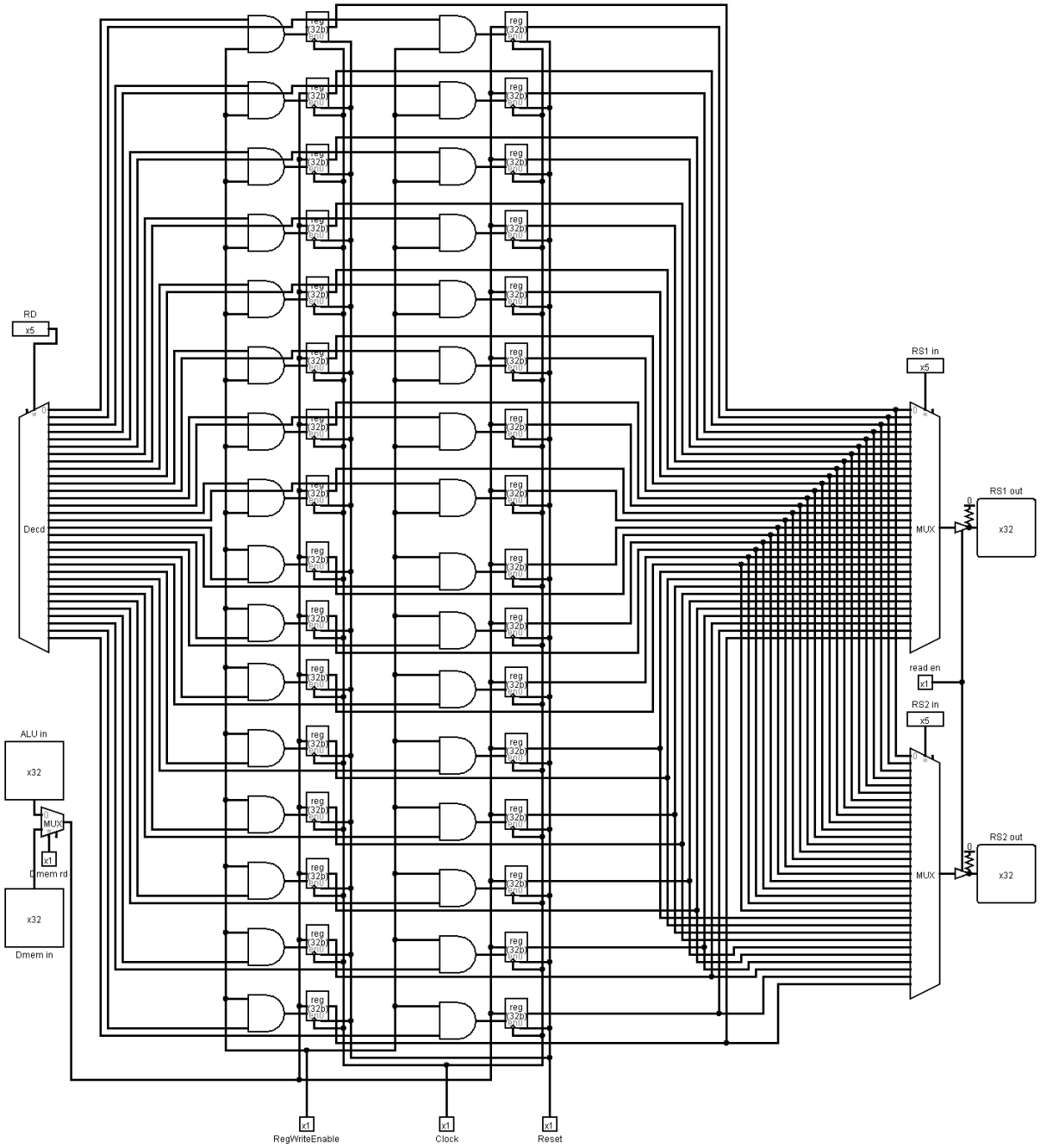
La parte centrale del diagramma è occupata da un contatore, composto da due elementi: uno a 3 bit e uno a 8 bit, entrambi pilotati da CLOCK e dotati di ingresso CLEAR. Il contatore a 3 bit funge da selettore di posizione o indice interno; il contatore a 8 bit è verosimilmente responsabile del conteggio generale o di un ciclo di memoria. L’uscita del contatore viene utilizzata per indirizzare la scrittura o per abilitare specifici percorsi nella logica combinatoria.

Il nodo principale in uscita dal percorso di controllo viene collegato a un blocco indicato come c_in / c_out, che appare come un modulo di gestione del carry o una logica di

trasferimento condizionale. Il segnale risultante raggiunge la linea che conduce al registro di uscita.

Il blocco OUTPUT, a 8 bit, è infine delimitato da una porta controllata tramite read enable. Solo quando il segnale di lettura viene attivato, il valore presente nella memoria interna viene trasferito verso l'esterno.

Register File



Lo schema rappresenta l'implementazione completa di un Register File a 32 registri × 32 bit.

A sinistra è presente un decoder, pilotato dall'indirizzo RD (5 bit), che individua quale dei 32 registri deve essere scritto. Le sue 32 linee di uscita vengono distribuite verticalmente e combinate con il segnale di RegisterWriteEnable, tramite porte AND poste in corrispondenza di ciascun registro. Questo approccio garantisce che solo il registro selezionato riceva effettivamente il segnale di scrittura al fronte di clock.

Ogni riga verticale della parte centrale rappresenta un registro a 32 bit, implementato come array di latch/flip-flop elementari (i blocchi “reg”). Le 32 colonne orizzontali rappresentano i bit; l’intersezione tra riga e colonna identifica il singolo flip-flop responsabile di un bit di un registro specifico.

Il percorso dati in ingresso presenta tre possibili sorgenti:
– ALU in (x32)
– Mem in (x32)
– Dmem in (x32)

La selezione tra queste avviene mediante un multiplexer dedicato, che instrada il valore verso la matrice dei registri.

Sul lato destro si trovano due complessi multiplexer 32->1, che producono le due uscite di lettura tipiche di un register file RISC: RS1 out e RS2 out. Ognuno seleziona uno dei 32 registri sulla base degli indirizzi RS1 e RS2 (anch’essi a 5 bit). La fitta rete di linee mostra esattamente le 32 × 32 connessioni necessarie perché ogni bit di ogni registro sia selezionabile.

In fondo compaiono il Clock, il Reset e la logica di abilitazione alla scrittura, che sincronizzano l’intero sistema.