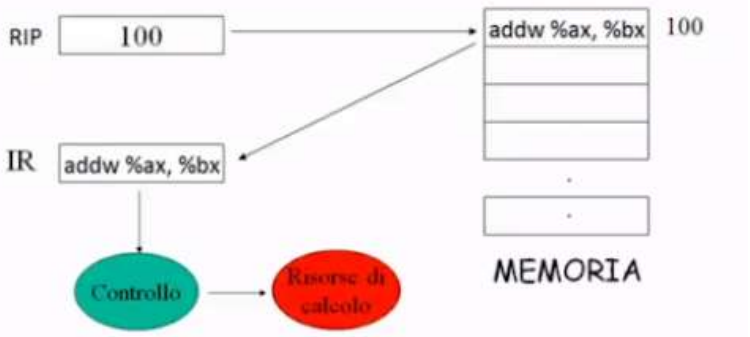
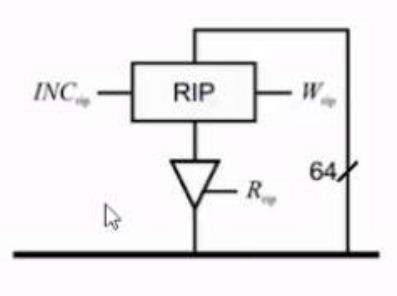
Lezione 17 Z64 3 + istruzioni 1

martedì 7 novembre 2023 15:27

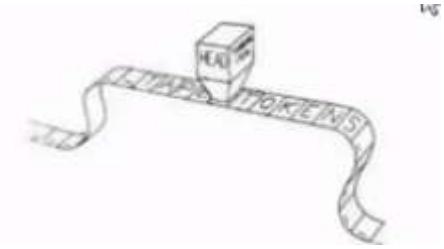
Andiamo a vedere come viene eseguita un'istruzione : fetch -> decode -> execute :



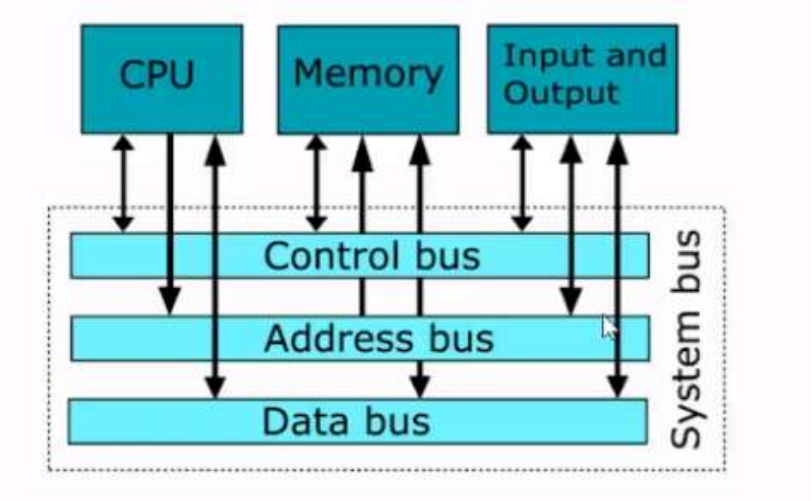
Ricordiamo che incrementare RIP di 1 significa spostare in avanti di 8 bit . Questo incremento di 8 in hardware viene realizzato cosi': dobbiamo fare una somma di 8 (terzo bit meno significativo) all'indirizzo precedente , quindi aumenta il numero di somme che si devono fare . Quindi usiamo dei ***registri contatori*** , ovvero registri che incrementano il contenuto dell'IP (usando dei Full-Adder) :



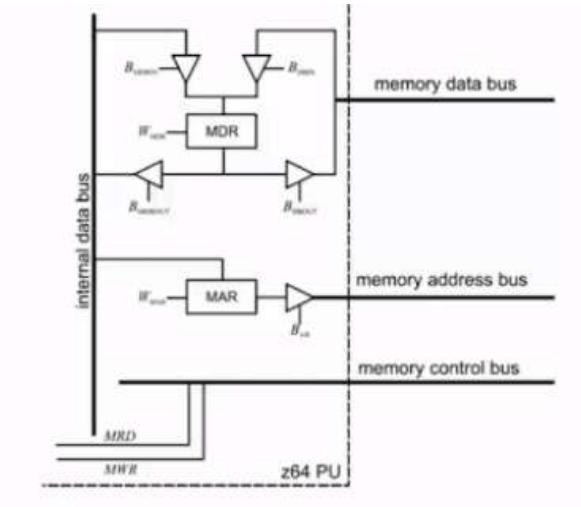
Dove INC è un segnale di controllo che permette di fare incremento del valore . Tornando ad architettura di Von Newmann , andiamo a focalizzarci sul registro MAR : è un registro che permette di scambiare informazioni tra processore e RAM , quindi nella fase di fetch prende l'indirizzo dell'istruzione da RIP . Si ha quindi che il segnale viaggia sul bus dati. Focalizziamoci sul RIP : indirizzo istruzione corrente , il processore dopo il fetch abilita il segnale di controllo : quindi RIP punta ad istruzione prossima . **Quindi RIP mantiene indirizzo istruzione corrente nella fase iniziale di fetch** . Per quanto riguarda la memoria : lo pensiamo come un nastro di n celle (contenenti i dati) , di dimensione fissata (8 bit -> 1 byte) . Ogni cella ha un numero e si parte dal numero 0 fino ad N-1 : ovvero l'indirizzo in memoria (spiazzamento all'interno del vettore) . Attenzione a contesto indirizzo e contesto dati .



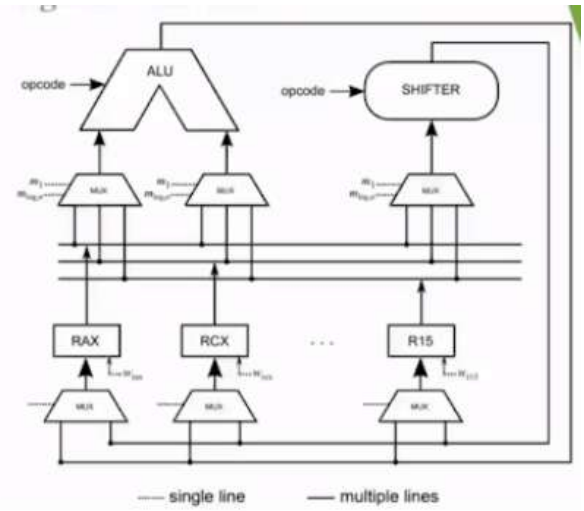
Vediamo ora come avviene lo scambio di info tra registri e processore : attraverso il **bus di sistema** (64 fili) , il quale si divide **control bus , address bus e data bus** :



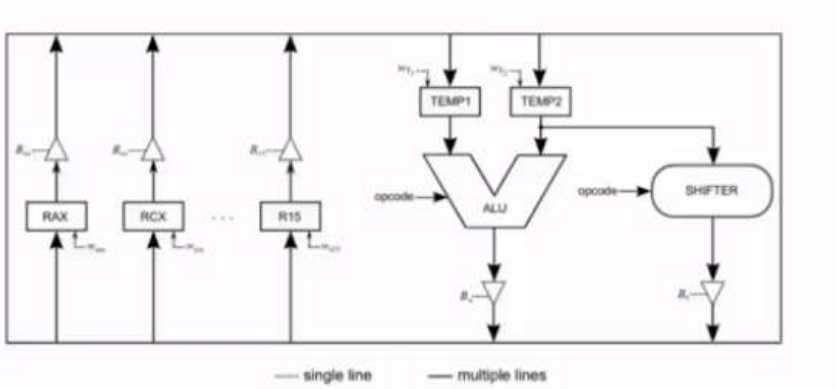
I primi sono fili che permettono comunicazione CPU - memoria . Nel secondo invece anche se si parla di architetture a 64 bit, si fermano a 48 bit (2^48 bit) . Vediamo ora come è realizzato :



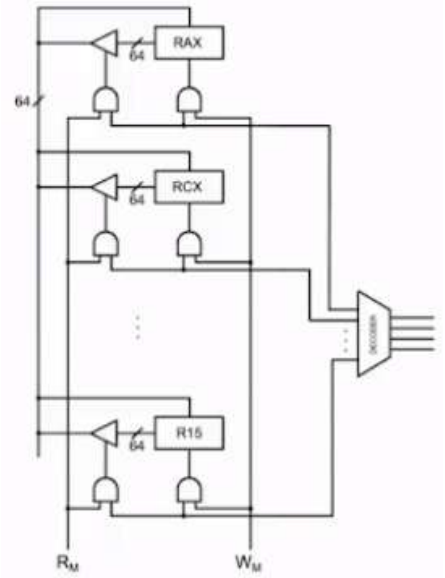
Notiamo il Wmar (write enable mar) : segnale di controllo che permette la scrittura del dato in memoria . Per quanto riguarda invece il buffer-tri-state si comporta come interruttore . Analogamente per MDR che ne ha 4 di buffer-tri-state (due in ingresso, due in uscita) . **Questa struttura di interconnessione, non è completa** in quanto manca quella che permette l'interconnessione tra registri e circuiti di calcolo :



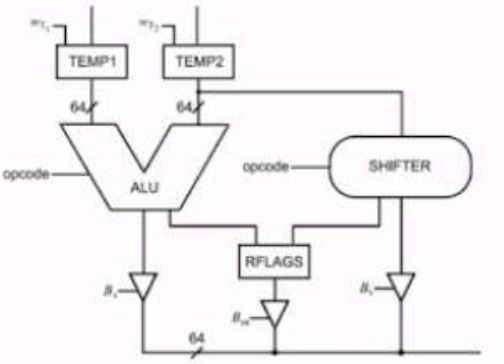
Con internal data bus non si riesce a trasferire dati (in modo stabile)verso ALU , quindi introduco un canale dedicato per questa comunicazione : ma c'è un problema : 16 mux , segnali di controllo aumentati , stesso registro scritto da alu e shifter !! Come risolviamo ? Uso registri tampone (2): vengono collegati al bus dati interno :



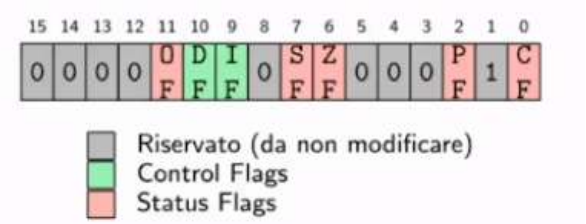
Attenzione alla fase nella quale la ALU fa operazioni (calcola dati errati) : o aggiungo opcode ad Alu oppure ignoro totalmente output , grazie al buffer-tri-state, il quale non permette di propagare queste informazioni errate . Per ogni registro general-purpose ci sono 2 segnali di controllo (W/R) , avendo cosi' 32 fili che escono da unità di controllo ed alu è ASF , quindi 32 segnali di ingresso : riorganizziamoli (disaccoppio W/R dai dati) : uniche due operazioni che posso eseguire sono W/R , quindi il numero dei registri scende a 16 : 4 segnali di controllo . Quindi in generale : da funzione che uscita a 32 bit a funzione che ha in uscita 6 bit (4 controllo + W + R) . Arriviamo cosi al **banco dei registri** : selezione un solo registro per eseguire un'operazione di W/R :



Ogni registro viene codificato in binario (4 bit) , ed inoltre hanno in ingresso due segnali di controllo (Rm e WM) , i quali vanno in and con l'uscita del decoder . Disaccoppiamento una linea di selezione registro da segnale di controllo che dice quale operazione eseguire su quel registro. Torniamo a parlare del **registro Flags** : registro che mantiene i bit di stato in uscita dalle reti iterative (shift ed ALU) e li memorizza.:



I dati vengono estrapolati dal registro flags usando la connessione con il data bus interno , la quale comunicazione (permesso/negazione) viene gestita dal buffer-tri-state. Com'è organizzato?



I bit di stato sono aggiornati da ALU e shifter per memorizzare le informazioni riguardanti l'ultima operazione eseguita , mentre quelli di controllo sono modificabili da programmatore per modificare le funzionalità del processore . C'è un bit settato ad 1 (posizione 1) , il quale bit verrà usato e collegato opportunamente per "creare"/interagire con valori costanti (costanti) . In dettaglio , ma attenzione al contesto :

- **carry** (CF): vale 1 se l'ultima operazione ha prodotto un riporto
- **parity** (PF): vale 1 se nel risultato dell'ultima operazione c'è un numero pari di 1
- **zero** (ZF): vale 1 se l'ultima operazione ha come risultato 0
- **sign** (SF): vale 1 se l'ultima operazione ha prodotto un risultato negativo
- **overflow** (OF): vale 1 se il risultato dell'ultima operazione supera la capacità di rappresentazione
- **interrupt enable** (IF): indica se c'è la possibilità di interrompere l'esecuzione del programma in corso
- **direction** (DF): modifica il comportamento delle operazioni su stringhe

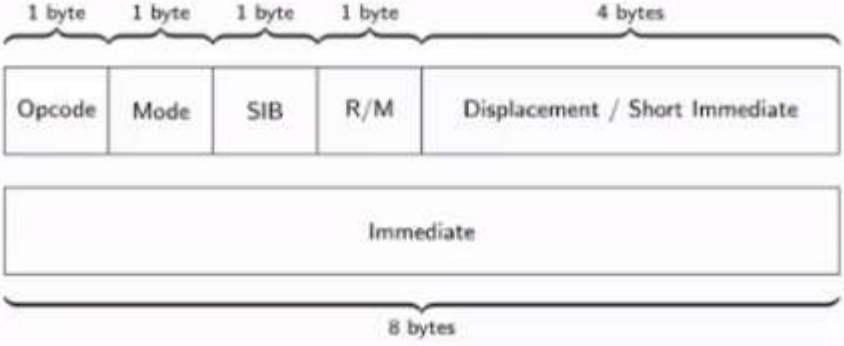
Sign lavora con il complemento a 2 . Gli ultimi due bit sono bit di controllo .

Andiamo a vedere come implementare le istruzioni nello z-64 :

Sono organizzate in otto classi:

- Classe 0: Controllo dell'hardware
- Classe 1: Spostamento dati
- Classe 2: Aritmetiche (su interi) e logiche
- Classe 3: Rotazione e shift
- Classe 4: Operazioni sui bit di FLAGS
- Classe 5: Controllo del flusso d'esecuzione del programma
- Classe 6: Controllo condizionale del flusso d'esecuzione del programma
- Classe 7: Ingresso/uscita di dati

Mentre per quello che riguarda il formato delle istruzioni macchina :



Il primo è un codice identificativo numerico che specifica una opportuna operazione (semantica

istruzione) , il secondo descrive il modo nel quale istruzione deve essere eseguita (vede opcode), il terzo serve a descrivere come vogliamo accedere in memoria , il quarto specifica se gli operandi sono registri di uso generale o qualche locazione di memoria, mentre l'ultimo si usa per gestire le costanti .