TEMPI MEDI E TEMPI TOTALI

	IF accesso mem.	ID register file	EX Alu	MEM accesso mem.	WB register file
LW	....	....	....	....	....
SW	....	....	....	....	
R	....	....	....		....
BEQ	....	....	....		
J	....	....			

quindi:

- LW = 2 MEM + 2 REG FILE + ALU
- SW = 2 MEM + 1 REG FILE + ALU
- R = 1 MEM + 2 REG FILE + ALU
- BEQ = 1 MEM + 1 REG FILE + ALU
- J = 1 MEM +1 REG FILE

SINGOLO CICLO

Ttotale = Tmedio * num_istruzioni

Tmedio = tempo lw

CICLO MULTIPLO

Ttotale=

- prendo sottofase che dura di piu (accesso a memoria)
- la moltiplico per il numero di sottofasi che ha ogni tipo di istruzione (ad esempio lw fa 5 fasi, quindi 5*4)
- per ogni tipo di istruzione (quindi AL, lw, Sw …) moltiplico il valore calcolato prima per il numero di volte che quella istruzione viene utilizzata
- sommo tutti questi valori e ottengo il T Totale

Tmedio = Ttot/num_istruzioni

CICLO VARIABILE

Ttotale =

- calcolare tempo impiegato da ogni tipologia di istruzione (R,lw,sw..)
- moltiplicare tempo totale di ogni tipologia di istruzione per il numero di istruzioni di quel tipo presenti
- Sommare tutti i valori

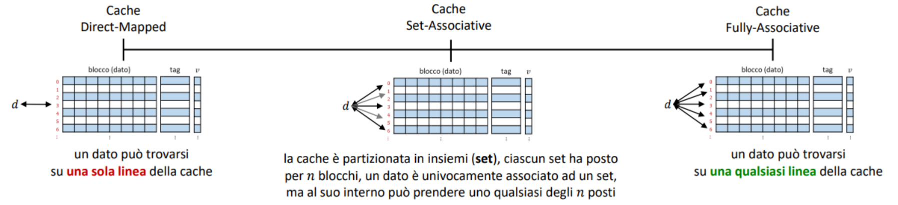
Tmedio = Ttotale / num_istruzioni

risingpirates.com

CACHE

RICORDA:

- **N(D)** = M(D) / B (in byte)
- **I(D)** = N(D) mod L
- **offset_blocco=m=**resto(N(D)) / 2^2
- **offse_parola =** Resto(N(d)) mod 2^2



MAPPATURA DIRETTA

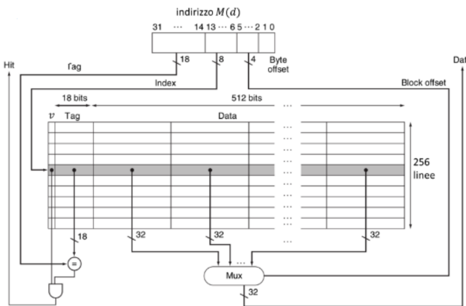
- **L** = Ddata / B
- **k** = $\log_2$(L)
- **m** = $\log_2$(B)-2
- **Tag** = 32 - (m + 2 + k)
- **Dtot** = L * (8B + TAG + 1) / 8

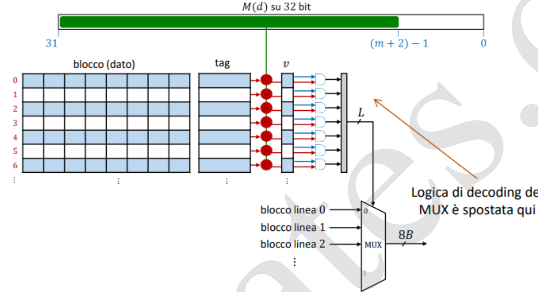
FULLY ASSOCIATIVE

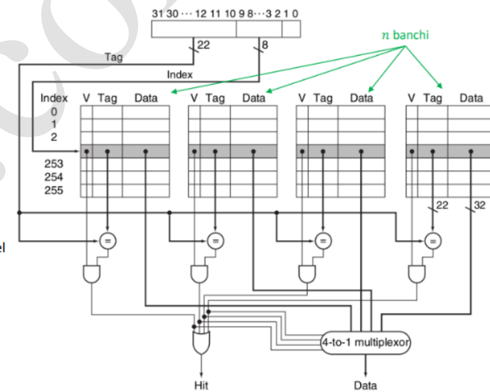
- **L** = 1
- **k** = 0
- **m** = $\log_2$ (B) - 2
- **Tag** = 32 - (m + 2)
- **Dtot** = L * (8B + TAG + 1) / 8

SET ASSOCIATIVE

- **L** = Ddata / n*B
- **k** = $\log_2$ (L)
- **m** = $\log_2$ (B)-2
- **Tag** = 32 - (m + 2 + k)
- **Dtot** = L * (8B + TAG + 1) / 8







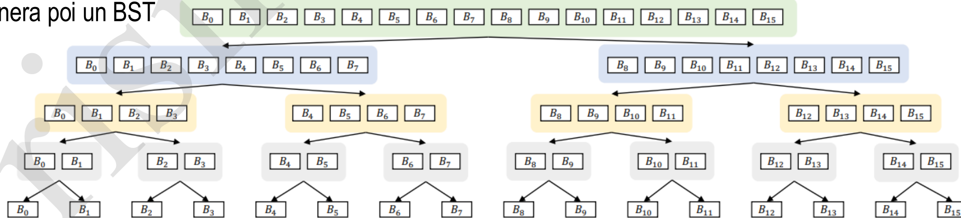
SOSTITUZIONE DEI BLOCCHI

1) *LRU*

- Si basa sulla tecnica degli use bits
- Per ogni set (riga) tiene traccia di qual è l'ultimo banco a cui abbiamo fatto accesso
- Permette anche di memorizzare quindi quello che abbiamo utilizzato per ultimo
- Se un banco ha piu di un blocco diventa complicato

2) *PSEUDO LRU*

- Utilizza un Binary Search Tree: si prendono tutti i blocchi presenti in un set e si numerano da b0 a bn
- Si genera poi un BST



- Ogni accesso al set, viene memorizzato nel BST in questo modo:
- Tra ogni diramazione si immagini un "bit di passaggio" che:
  - Viene settato a **zero** se andiamo a **sinistra**
  - Viene settato a **uno** se andiamo a **destra**
- per scrivere in quello utilizzato meno di recente →basta andare nella direzione completamente opposta e scrivere nel blocco in cui arriviamo (man mano che si passa attraverso l'albero anche in scrittura ovviamente bisogna settare il bit di passaggio)
- con n vie servono n-1 bit!

risingpirates.com

CPU A PIPELINE

CRITICITA' STRUTTURALE

senza la duplicazione dei componenti, varie istruzioni possono avere conflitti sulle unita' della CPU. Necessario

- 1 ALU → 3 ALU (BTA, Operazioni, PC+4)
- 1 RF → Non crea problemi strutturali
- 1 MEM → 2 MEM (Istruzioni, Dati)

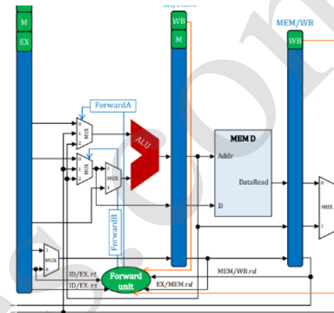
CRITICITA' DI DATO

1) HAZARD A/L

- Gestione Software: inserimento NOP (2 dopo ogni istruzione) oppure inserimento di istruzioni che non interferiscono e sono indipendenti dalla loro posizione nel codice
- Gestione Hardware: Forwarding → i due operandi possono ottenere valori diversi in base ai segnali di controllo che la Hazard Unit produce:

  PER OPERANDO A (Forward A) :
  - 00 → mantieni RS
  - 01 → prendi il dato da MEM/WB
  - 10 → prendi il dato da EX/MEM

  PER OPERANDO B (Forward B) :
  - 00 → mantieni RD / IMMEDIATE (scelto nel MUX prima)
  - 01 → prendi il dato da MEM/WB
  - 10 → prendi il dato da EX/MEM



2) HAZARD LW

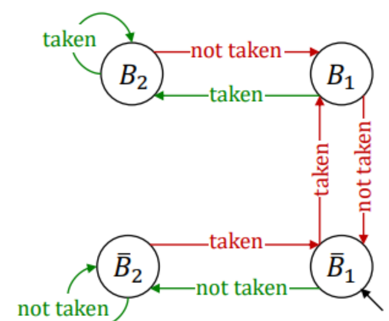
- Se ho una LW che carica un dato in un registro e subito dopo ho due istruzioni che lo usano allora ho una criticità di dato per LW
- seconda istruzione che segue la lw, se usa il dato si risolve con il forward di prima, perche' il dato è già pronto in memoria, va solo forwardato all'indietro
- la prima istruzione che segue la lw invece non ha ancora il dato pronto → Hazard unit da sola non può risolvere. Quindi:

  1) Si rileva un hazard in ID
  2) si scrive nello stesso ciclo di clock isegnali di controllo di una NOP in ID/EXE
  3) PCWrite →0 (devo rifetchare questa istruzione)
  4) Nel ciclo successivo:
     - IF e ID vengono rieseguite (perche' non ho aggiornato il PC)
     - EX svolge una nop (forzata nei segnali di controllo precedentemente)
     - MEM e WB continuano senza problemi

CRITICITA' DI CONTROLLO

- Approccio software: 3 Branch Delay Slots → sotto ogni beq metto 3 slot vuoti e in ogni BDS metto delle istruzioni che non generano conflitti.
- Approccio Hardware: Anticipazione del salto:
  - Anticipo il calcolo del BTA e la verifica della branch nella fase di ID.
  - scommetto sempre su NOT TAKEN ovvero, dopo la branch comunque faccio andare l'istruzione che c'e' subito dopo, se invece la branch è taken allora pago un ciclo di clock.
  - se il salto viene preso, devo flushare il registro IF/ID al cilclo di clock subito dopo, e ovviamente aggiornare il PC con il BTA.
  - *NB: Potrebbero esserci Data Hazard che vengono gestiti con delle tecniche di forwarding uguali a quelle per A/L*
- la tecnica sopraindicata scommette sempre su NOT TAKEN, se vogliamo qualcosa di dinamico:
  - Dynamic Branch Prediction ad 1 bit (cpu scommette che l'esito del salto è uguale a quello precedente. Problema? Loop innestati)
  - Dynamic Branch Prediction a 2 bit (cpu scommette su esito tramite una FSM)

Nome dello stato	Codifica	Uscita (prediction)
$\bar{B}_1$	00	0
$\bar{B}_2$	01	0
B_1	10	1
B_2	11	1



CODICI DI RILEVAMENTO E CORREZIONE ERRORE

Distanza **di Hamming** = rappresenta il numero di bit in cui due parole di codice binario differiscono

CODICE A RIPETIZIONE

- Chiamiamo d la distanza di hamming un codice a ripetizione ripete ogni bit d-1 volte
- Quindi 1 0 0 1 con d=4 codificato a ripetizione diventa 1**111** 0**000** 0**000** 1**111**
- Questo codice permette di
  - **RILEVARE** $d-1$ errori
  - **CORREGGERE** $\frac{(d-1)}{2}$ errori

CODICE DI HAMMING

- Codice di hamming di base ha d=3
- In base al numero di bit che ho da codificare avrò x bit di parità
- Come funziona:
  - Nelle posizioni delle potenze di due devo inserire dei bit di parità dove:
    - P1 = è bit di parità considerando i bit " 1 si 1 no" a partire dal bit in posizione 1
    - P2 = è bit di parità considerando i bit " 2 si 2 no " a partire dal bit in posizione 2
    - P4 = è bit di parità considerando i bit " 4 si 4 no " a partire dal bit in posizione 4
    - Etc
  - Una volta calcolati questi bit, vengono inseriti nelle rispettive posizioni all'interno della sequenza da codificare
- Con una codice di hamming a d=4 bisogna aggiungere un bit di parità aggiuntivo.
  - Semplicemente, alla fine della sequenza di bit codificata si aggiunge un bit di parità finale, calcolato su TUTTA la sequenza codificata fino ad ora.
  - Il bit di parità aggiuntivo durante il calcolo di correttezza della sequenza NON deve essere considerato.

Identificare e correggere l'errore

- Data una sequenza codificata in hamming, per poter identificare il bit sbagliato è necessario calcolare ECC = Sommo la posizione dei bit di parità che mi risultano sbagliati. Se p1 e p2 mi risultano sbagliati, significa che il mio bit sbagliato sarà il numero 3.
- Se considero anche il bit di parità allora:
  - Se ECC=0 e bit aggiuntivo corretto → non ci sono errori
  - Se ECC=0 e bit aggiuntivo sbagliato → il bit di parità aggiuntivo è sbagliato
  - Se ECC>0 e bit aggiuntivo corretto → doppio errore impossibile da correggere
  - Se ECC>0 e bit aggiuntivo sbagliato → singolo errore che si può correggere (come definito prima)

risingpirates.com

EXCEPTIONS

- Exception = Interruzione causata dalla CPU internamente o esternamente
- Interrupt = Interruzione causata da qualcosa di esterno alla CPU
- Exception può essere di due tipi:
    1) Istruzione non riconosciuta (in fase di ID)
    2) Overflow aritmetico (in fase di EXE)
- In genere bisogna trovare
    - Causa eccezione (tipo di istruzione)
    - Offending instruction (chi ha causato cosa)
    - Risposta all'eccezione tramite exception handler
    - Terminazione o ripresa del programma
- **EPC** = Exception Program Counter → tiene indirizzo dopo offending instruction ovvero PC+4 che è già presente in IF/ID per poter dopo tornare alla normale esecuzione
- Per gestire l'eccezione poi è possibile utilizzare:
    - **REGISTRO CAUSA**
    - **INTERRUPT VECTOR TABLE**

REGISTRO CAUSA

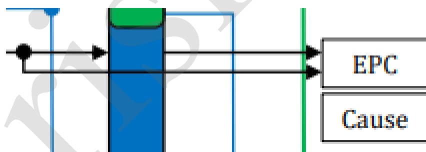
- Si memorizza il codice di errore nel registro causa
- Si fa un salto all'indirizzo 0x800000180 dove è presente la prima istruzione dell'**exception handler**
- L' handler legge il registro causa e agisce di conseguenza
- Intanto l'EPC si è memorizzato PC+4

INTERRUPT VECTOR TABLE

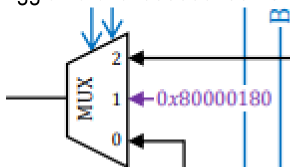
- In base all'exception, l'hardware fa un salto ad un indirizzo specifico che ha la soluzione al problema

DATAPATH

- Anche il datapath subisce delle modifiche:
    - Gestione delle due tipologie di exception da parte della CU:
        - **UNKNOWN OPERATION:**
            - Flush di offending instruction in ID/EXE
            - Flush di istruzione successiva in IF/ID
        - **OVERFLOW**
            - Flush di offending instruction in EX/MEM
            - Flush di istruzione successiva in ID/EX
            - Flush istruzione successiva ancora in IF/ID
    - Inserimento di due registri: EPC e CAUSA



    - Aggiunta di 0x800000180 nel MUX del PC

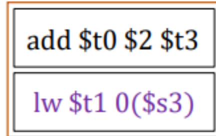


INSTRUCTION LEVEL PARALLELISM

- Piu pipeline in una CPU, k-vie = numero di pipeline
- Problema → alcune istruzioni non possono essere lanciate insieme
- Soluzione → Preparo un issue packet, se l'issue packet non crea problemi, ottimo altrimenti **Roll Back (annullamento effetti istruzioni)**

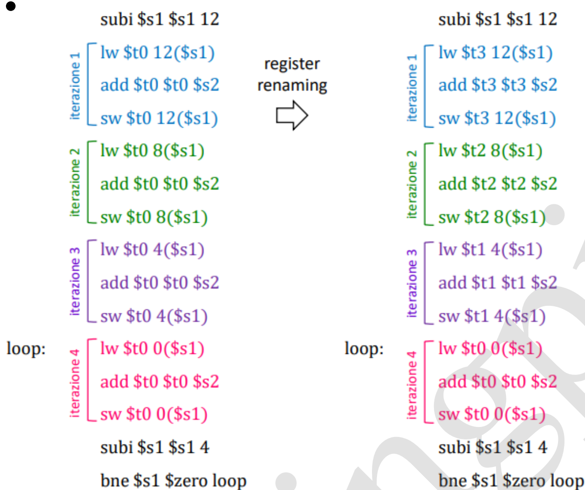
MULTI ISSUE STATICA

- si genera un issue packet strutturato nel seguente modo:



- 32 bit → operazione Aritmetico Logica / Branch / J
- 32 bit → operazione di memoria lw o sw

- Ovviamente si formano molti piu hazard che deve gestire il **compilatore**, purtroppo quindi in molti casi alcune issue packet si riempiono di NOP
- Una tecnica utilizzata per gestire i cicli, che di base genererebbero tantissimi hazard di controllo, è quella del **loop unrolling**
  - Compilatore analizza il codice e vede che il ciclo viene eseguito almeno x volte
  - Srotola il ciclo: prende quelle istruzioni e le riscrive in modo sequenziale x volte
- Per quanto bella questa tecnica sia, così e basta genera moltissimi hazard in piu, per questo si utilizza il **register renaming**
  - Si prende ogni "blocco" di quel loop e si riscrive con dei registri che non possono generare hazard con il blocco precedente
  - NB: gli hazard interni tra i blocchi rimangono e vengono gestiti sempre dal compilatore e dalla CPU nel datapath!
-



Con loop unrolling e register renaming

A/L o branch	Accesso a memoria
subi $s1 $s1 12	lw $t3 0($s1)
	lw $t2 8($s1)
add $t3 $t3 $s2	lw $t1 4($s1)
add $t2 $t2 $s2	lw $t0 0($s1)
add $t1 $t1 $s2	sw $t3 12($s1)
add $t0 $t0 $s2	sw $t2 8($s1)
subi $s1 $s1 4	sw $t1 4($s1)
bne $s1 $zero loop	sw $t0 4($s1)

$IPC_{max} = 2, IPC_{min} = 1$

IPC empirico: $IPC_e = \frac{num.di\ istruzioni}{num.cilci\ di\ clock} = \frac{15}{8} = 1{,}875$

percentuale di efficienza: $\rho = \frac{IPC_e - IPC_{min}}{IPC_{max} - IPC_{min}} = 1{,}875 - 1 = 0{,}875$

Senza loop unrolling

A/L o branch	Accesso a memoria
	lw $t0 0($s1)
add $t0 $t0 $s2	
subi $s1 $s1 4	sw $t0 0($s1)
bne $s1 $zero loop	

$IPC_e = \frac{5}{4} = 1{,}25$

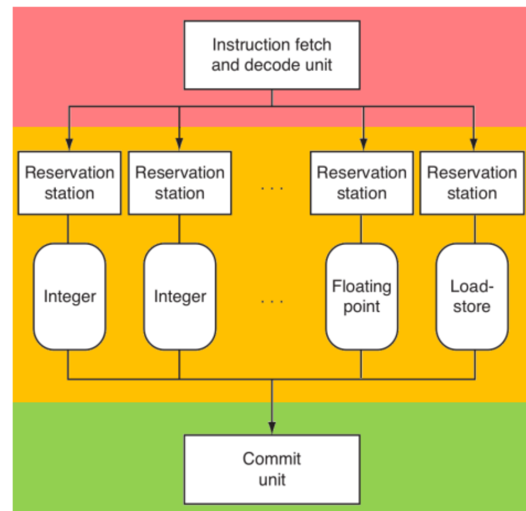
$\rho = \frac{IPC_e - IPC_{min}}{IPC_{max} - IPC_{min}} = 1{,}25 - 1 = 0{,}25$

- Applicando il loop unrolling otteniamo un miglioramento del 60%

MULTI ISSUE DINAMICA

- Il compilatore riorganizza il codice e poi viene gestito a livello hardware

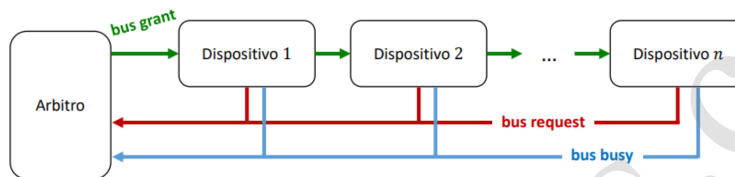
- Istruzioni vengono prelevate una alla volta in base a come sceglie il compilatore.
- Quando vengono fetchate, la reservation station mantiene un **record** formato da **operazione + operandi**
- Ogni unità funzionale lavora indipendentemente e in parallelo alle altre
- La commit unit è un buffer che riordina le istruzioni e poi fa un commit
- Se è presente un **hazard**, il valore dell'operando non viene copiato e l'ingresso dell'istruzione nell'unità funzionale viene posto in attesa.
- L'istruzione quindi "prenota" l'ingresso ed entra appena possibile
- Quando può entrare, entra e viene eseguita. Successivamente, il valore viene messo in commit ed inviato a chi ne ha bisogno.



risingpirates.com

INPUT / OUTPUT

- **Bus dati =** bus dove vengono trasferiti i dati utili alla CPU
- **Bus Controllo =** bus dove vengono trasferite le informazioni riguardanti il controllo del BUS
- **Bus Indirizzi =** bus dove vengono trasferiti gli indirizzi di memoria
- Chi utilizza questi bus deve sapere quando parlare e quando ascoltare, come si fa?
  - **BUS SINCRONO** = il bus contiene un suo clock e si utilizza per sincronizzarsi.
    - Bus velocissimi ma molto complessi con vincoli di frequenza e necessita che siano corti per evitare ritardi
  - **BUS ASINCRONO** = i bus utilizzano un handshake
    - Bus piu lenti ma che possono essere anche lunghi
- Come si fa a sapere chi ha il diritto ad utilizzare il bus in un determinato momento? → **Arbitraggio del BUS**
  - **DAISY CHAIN**
    - I devices sono messi in ordine di priorità, da quello con meno priorità a quello con maggiore priorità
    - Un device fa una **bus request**
    - L'arbitro attiva il segnale di **bus grant** che passa attraverso tutti i device fino ad arrivare al device che lo ha richiesto ed esso smette di mandare avanti il segnale di bus grant.
    - Problema = ritardi di propagazione, se un nodo collassa tutto collassa, non c'è equità



  - **DAISY CHAIN A RICHIESTE INDIPENDENTI**
    - Uguale al daisy chain ma ogni device ha la sua linea dedicata per fare bus request e ricevere il bus grant
    - Piu fairness ma molto piu complesso da gestire ed implementare
  - **PRIORITA STATICA E ALTRI PROTOCOLLI**
    - Per arbitraggio del BUS si può utilizzare una priorità statica oppure semplicemente definire altri protocolli per rendere piu fair il meccanismo.
- Per fare un accesso ad un device, è possibile agire tramite due tecniche:
  - **MEMORY MAPPED**
    - Si maschera l'accesso alla periferica come se fosse un accesso a memoria, mappando quindi l'indirizzo della periferica ad un indirizzo di memoria .
    - Questo indirizzo di memoria è linkato direttamente al buffer di lettura e scrittura della periferica
  - **ISTRUZIONI SPECIALI**
    - Si definire un' ISA in grado di supportare la diretta comunicazione
- Un device è strutturato nel seguente modo:
  - REGISTRO DATI = buffer temporaneo per la lettura e la scrittura dei dati sul bus (usa bus di dato)
  - REGISTRO DI STATO = codici che definiscono lo stato attuale della periferica (usa bus di controllo)
  - CONTROLLER = logica di controllo per accedere al bus e trasferire i dati (usa bus indirizzi)
- Come si coordina l'attività della CPU con quella di una periferica all'interno di un programma
  - POLLING = anziché mandare in stallo la CPU si controlla periodicamente il registro di stato delle periferiche
  - INTERRUPT = la periferica manda un segnale di interrupt alla CPU che interrompe il suo flusso e poi viene gestita come una eccezione. Per interrupt a livelli di priorità diversi si utilizza una interrupt mask

PRESTAZIONI

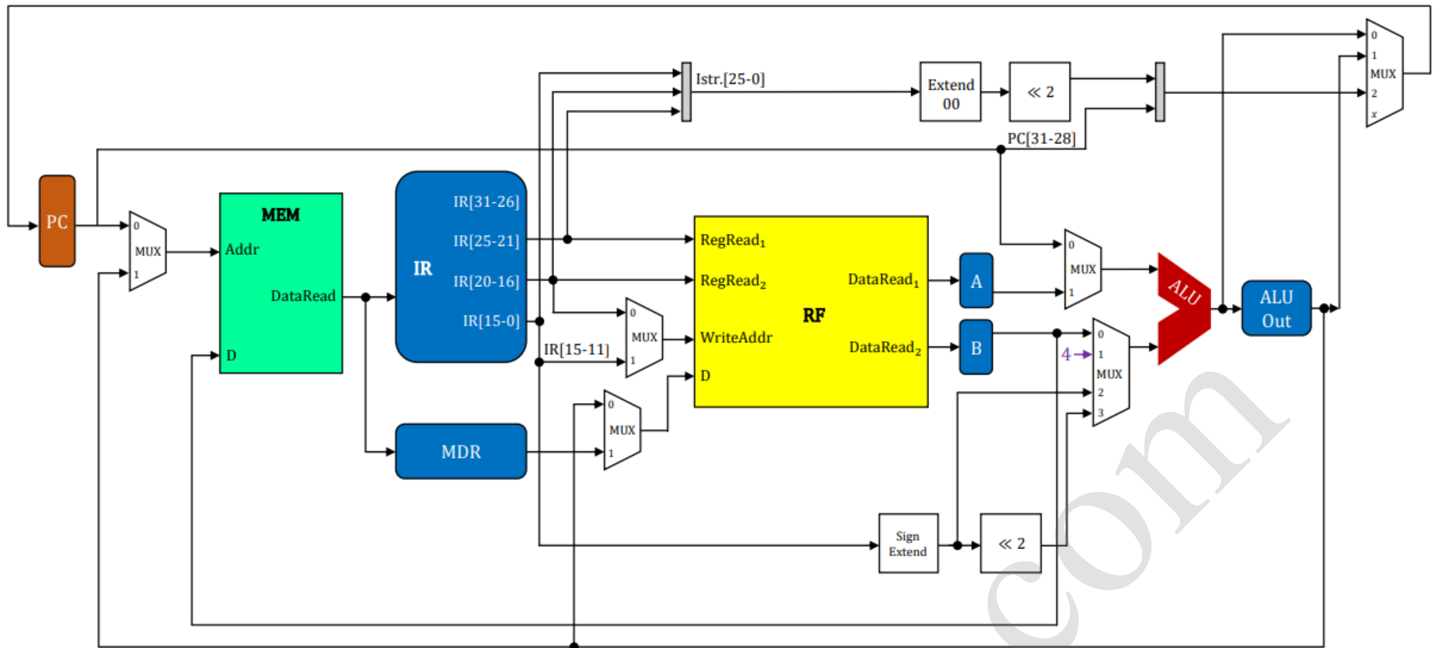
- CPI = Numero di istruzioni per ciclo di clock
- P = programma
- n_i = numero di istruzioni di tipo i in p
- n = numero totale di istruzioni
- N = numero di tipi di istruzione
- f_i = Frequenza dell'istruzione i nel programma p
- CPI (negato) = CPI medio del programma p
- T (negato) = Tempo di attesa medio del programma p
- Tck = 1/(Frequenza CPU)

$$\overline{CPI} = \sum_{i=1}^{N} CPI_i \times f_i$$

$$\overline{T} = \overline{CPI} \times n \times T_{ck}$$

risingpirates.com

CPU A CICLO MULTIPLO



ALUSrcA	Selezione del primo operando ALU: PC (0) registro A (1)
ALUSrcB	Selezione del secondo operando ALU: Registro B (00), costante 4 (01), SignExt($OFFSET$) (10), SignExt($OFFSET$) $\times$ 4 (11)
IorD	Selezione per il campo Addr nel modulo memoria: indirizzo istruzione (0), indirizzo dato (1)
PCSrc	Selezione di quale indirizzo mandare al PC: risultato della ALU (00), contenuto di ALUOut (01), indirizzo della jump (10), (11 non usato)
RegDest	Selezione per il campo WriteAddr del RF: rt (0) rd (1)
MemToReg	Selezione del dato presentato in scrittura al RF: contenuto di ALUOut (0), contenuto di MDR (1)
ALUCtrl	Selezione della modalità di operazione della ALU (analogo a CPU singolo ciclo)

Ricorda:

Opcode	AluCtrl
Tipo R, rimanda a funct	10
sw, somma	00
lw, somma	00
beq, sottrazione	01

PCWrite	Scrittura del Program Counter
Branch	Se posto a 1 abilita la scrittura del PC quando il bit di zero della ALU vale 1, da usare per i salti condizionati
IRWrite	Scrittura dell'Instruction Register
RegWrite	Scrittura del Register File
MemWrite	Scrittura della memoria
MemRead	Lettura della memoria

risingpirates.com