Architettura degli Elaboratori

12 gennaio 2016

Indice

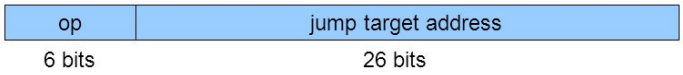
1 Istruzioni MIPS Assembler

1.1 Classi di istruzioni

- R-type, istruzioni aritmetico-logiche, come SUB, ADD

- I-type, istruzioni di tipo immediato, come LOAD, STORE, BRANCH, ADDi, SUBi

- J-type, istruzioni di salto, come JUMP



1.2 ISA - Istruction Set Architecture

Queste CPU devono essere in grado di eseguire: Istruzioni R-type:

- ADD
- SUB
- SLT - Set on Less Then
- AND
- OR

Istruzioni I-type:

- LW - Load Word
- SW - Store Word
- BEQ - Branch If Equal
- BNE - Branch if Not Equal
- ADDi - ADD Immediate
- SUBi - SUB Immediate

Istruzioni J-type:

- J - Jump

1.3 Scrivere sintassi e semantica delle operazioni MIPS BNE e SLT

L'istruzione BNE, branch if not equal, è di tipo immediate, ovvero I-Type. I primi 6 bit sono del campo operation, poi 5 bit del campo rs, altri 5 bit del campo rt e successivamente 26 bit per la costante immediata. Questa istruzione confronta i due registri rs e rt e se sono diversi salta all'istruzione indicata dal campo immediato (che è PC-relative).

L'istruzione SLT invece, set on less then, è di tipo R-Type. I primi 6 bit sono sempre dedicati al campo operation, i prossimi 15 bit sono quelli di 3 registri (da 5 bit ciascuno – rs,rt,rd) poi vi sono altri 5 bit che sono del campo shmat (shift amount, diversi da zero sono in caso di shifting) e gli ultimi 6 bit sono del campo funct (che determina il tipo dell'operazione). Questa istruzione confronta i due registri rt e rd e se rt $<$ rd imposta rs a 1, altrimenti a 0.

1.4 Scrivere la semantica dell'istruzione assembler BEQ. Se all'indirizzo di memoria 2000FC00 è presente l'istruzione beq 5,6,00A0 (indirizzo e costante espressi in esadecimale), quale sarebbe l'indirizzo target nel caso in cui il branch fosse taken?

L'istruzione BEQ, branch if equal, è di tipo immediate, ovvero I-Type. I primi 6 bit sono del campo operation, poi 5 bit del campo rs, altri 5 bit del campo rt e successivamente 26 bit per la costante immediata. Questa istruzione confronta i due registri rs e rt e se i contenuti sono uguali salta all'istruzione indicata dal campo immediato (che è PC-relative). Quindi se l'istruzione si trova all'indirizzo 2000FC00, ed è taken, l'indirizzo di salto sarà 2000FC00 + 00A0 = 2000FCA0.

1.5 Scrivere sintassi, semantica e formato delle seguenti istruzioni assembler: j, subi

L'istruzione Jump, salto incondizionato, aggiorna il PC con un nuovo indirizzo. L'istruzione j è, appunto, di tipo J-type, ed è formata da: 6 bit del campo operation e 26 bit per l'indirizzo (PC-relative). Per ottenere l'indirizzo di Jump devo: shiftare di 2 a sinistra l'indirizzo a 26 bit, e concatenare i primi 4 bit del PC a questi 28 bit appena ottenuti. Ottengo cosi un indirizzo a 32 bit da salvare nel PC.

L'istruzione SUBi, SUB immediate, è di tipo I-type. È formata da:

- 6 bit di operation
- 5 bit del registro sorgente (rs)
- 5 bit del registro destinazione (rt)
- 16 bit per la costante immediata, che rappresenta il secondo operando

La costante viene estesa a 32 bit quindi sommata ai 32 bit del contenuto del registro rs definito dai 5 bit di ingresso al Register File.

2 CPU Singolo Ciclo

2.1 Principi di progetto, svantaggi e vantaggi

La CPU singolo ciclo deve essere in grado di eseguire una le seguenti istruzioni: lw, sw, add, sub, slt, and, or, beq, jump in un solo ciclo di clock. Quindi ogni istruzione (che è composta da più sub-attività) deve essere eseguita in un unico ciclo. Il clock va dimensionato in corrispondenza dell'istruzione che richiede più tempo di elaborazione. Questo però comporta che le istruzioni più corte, che impiegano meno tempo del periodo del clock vengano penalizzate (essendo il clock a tempo fisso). Se ad esempio l'istruzione lw (load) richiede 5 sub-attività che impiegano 8 ms in totale e tutte le altre operazioni impiegano un tempo minore, il clock andrà dimensionato a minimo 8 ms, penalizzando, ad esempio, l'istruzione j (jump) che impiega solo 3 subs e 4 ms. I restanti 4 ms vengono sprecati dovendo la CPU attendere il nuovo ciclo di clock per iniziare l'istruzione successiva. Inoltre dovendo svolgere in un'unica istruzione più attività dello stesso genere (es. due somme, una per l'indirizzo di memoria e uno per incrementare il PC) nello stesso ciclo, siamo costretti a replicare le risorse che svolgono in questo caso la somma, in quanto la risorsa è già impegnata nella prima operazione quando è il momento di calcolare anche la seconda. Lo svantaggio principale di questa CPU è che (nel caso reale) dovendo svolgere operazioni FP, che richiedono molto tempo di elaborazione, il ciclo di clock venga allungato a dismisura, rallentando drasticamente i tempi di risposta.

3 CPU Multiciclo

3.1 Tabella istruzioni/cicli

Passo	R-Type	LW/SW	Branch	Jump
Fetch dell'istruzione + incremento PC	IR = Mem[PC] PC = PC + 4	*carico l'istruzione corrente* *anticipo dato che ALU è libera*		
Decode	A = RF[IR[25-21]] B = RF[IR[20-16]] ALUOut = PC + (sign_extend(IR[15-0]) << 2	*carico in A il primo registro (rs)* *carico in B il secondo registro (rt)* *calcolo l'indirizzo target del branch (anticipo, ALU libera)*		
Termino branch/jump + esecuzione op + calcolo indirizzo LW/SW	ALUOut = A op B	ALUOut = A + sign_extend(IR[15-0]) *calcolo indirizzo LW/SW*	If(A == B) then PC = ALUOut *Dove ALUOut è l'indirizzo del branch, calcolato il ciclo prima*	PC = PC[31-28] || (IR[25-0] << 2) *concateno i primi 4 bit del PC alla costante shiftato a sinistra di 2 (26 -> 28). Quindi 4 + 28 = 32 bit, indirizzo di j*
Accesso alla memoria LW/SW + termino R-Type	RF[IR[15-11]] = ALUOut	Load: MDR = Mem[ALUOut] Store: Mem[ALUOut] = B		
Termino LW		Load: RF[IR[20-16]] = MDR		

3.2 Principi di progetto, svantaggi e vantaggi

La CPU multiciclo deve essere in grado di eseguire le stesse operazioni di quella a singolo ciclo, ma in un modo ottimizzato. Questa CPU non dovrà più eseguire un operazione intera nello stesso ciclo di clock, ma piuttosto dovrà ad ogni ciclo terminare una delle sotto attività richieste dall'istruzione principale. Per ottenere questo risultato dobbiamo introdurre nuovi registri e nuovi controlli, che tengano traccia del punto di esecuzione, ovvero che riescano a capire e tenere memorizzato che passi (subs) abbiamo già svolto. In questo caso non abbiamo più bisogno di un ciclo di clock lungo come l'istruzione che impiega più tempo, ma ci basterà un ciclo di clock che basti a terminare la sotto operazione più costosa in termini di tempo. Oltre al clock ridotto, inoltre, non abbiamo nemmeno più il problema della replicazione esagerata delle risorse, infatti nel caso di prima, ovvero quando nella stessa istruzione vi sono due operazioni simili (due somme), nel primo ciclo completo la prima somma, e nel ciclo successivo la seconda, evitando così di replicare risorse.

3.3 Passi di esecuzione delle istruzioni nella CPU multiciclo

LOAD - 5 cicli

1) Carico nel IR il contenuto della memoria indicato da PC e aggiorno il contatore di PC (PC = PC + 4)
2) Carico nell'elemento di memoria A il contenuto del primo registro (rs) [e in B il contenuto del secondo. Nello stesso ciclo calcolo l'indirizzo del branch]
3) Calcolo l'indirizzo da cui leggere i dati per scriverli nel registro. Quindi A + offset
4) Carico nel MemoryDataRegister il contenuto della memoria indicato dall'indirizzo calcolato al ciclo precedente
5) Carico nel registro rt il contenuto di MDR

STORE - 4 cicli

1) Carico nel IR il contenuto della memoria indicato da PC e aggiorno il contatore di PC (PC = PC + 4)
2) Carico nell'elemento di memoria A il contenuto del primo registro [e in B il contenuto del secondo. Nello stesso ciclo calcolo l'indirizzo del branch]
3) Calcolo l'indirizzo in cui scrivere i dati letti dal registro il cui contenuto è in B
4) Scrivo nella memoria (all'indirizzo calcolato al ciclo precedente) il contenuto di B

BEQ, BNE - 3 cicli

1) Carico nel IR il contenuto della memoria indicato da PC e aggiorno il contatore di PC (PC = PC + 4)
2) Carico nell'elemento di memoria A il contenuto del primo registro e in B il contenuto del secondo
3) Se il caso è taken (A==B / A!=B) imposto il valore di PC all'indirizzo calcolato dall'ALU nel ciclo precedente

ADD, SUB - 4 cicli

1) Carico nel IR il contenuto della memoria indicato da PC e aggiorno il contatore di PC (PC = PC + 4)
2) Carico nell'elemento di memoria A il contenuto del primo registro e in B il contenuto del secondo. [Nello stesso ciclo calcolo l'indirizzo del branch]
3) Calcolo il risultato l'operazione (ADD/SUB) tra il contenuto di A e B
4) Scrivo nel registro destinazione (rd) il risultato proveniente dall'ALU dal ciclo precedente

3.4 La CPU multiciclo vista a lezione utilizza per l'esecuzione delle istruzioni SUB e LW rispettivamente 4 e 5 cicli di clock. Descrivere brevemente cosa accade nei vari cicli per le due istruzioni

L'operazione SUB, sottrazione, è di tipo aritmetico-logico, quindi R-type. Al primo ciclo carico nel Istruction Register il contenuto della memoria indicato dal PC e nello stesso ciclo incremento il valore di PC (PC = PC + 4). Nel ciclo successivo carico nell'elemento di memoria A il contenuto del Register File indicato dall'Istruction Register in base ai 5 bit del primo suo ingresso (rs) e svolgo lo stesso compito anche per l'elemento di memoria B (scrivo in B rt). In questo stesso ciclo calcolo l'indirizzo del branch nel caso fosse taken (passo di anticipo). A questo punto ho i due addenti dell'operazione negli elementi A e B, e faccio eseguire all'ALU l'operazione richiesta (SUB). A questo punto scrivo nel registro identificato da rd il risultato dell'ALU.

L'istruzione LW, Load Word, carica in un registro il contenuto di una cella di memoria. E' di tipo I-type ed impiega 5 cicli di esecuzione. Al primo ciclo (1) carico l'istruzione corrente, indicata dal PC. AL ciclo successivo (2) carico nell'elemento di memoria A il contenuto del rispettivo registro (rs). Nel ciclo successivo (3) calcolo l'indirizzo dal quale prendere i dati (determinato dalla somma di A + il campo immediate, che è l'offset). Al ciclo successivo (4) carico nel Memory Data Register il contenuto della cella di memoria calcolata al ciclo precedente. All'ultimo ciclo (5) carico nel registro (rt) il contenuto dell'MDR, caricato precedentemente con il contenuto della cella di memoria desiderata.

4 Hardware

4.1 Register File

Il Register File, contenuto nel datapath e formato da 32 registri (ognuno dei quali è a 32 bit, formati a loro volta da 32 flip-flop) è un elemento di memoria che contiene, ad esempio, gli addendi di un'operazione aritmetico-logica. Deve permettere la lettura di 2 registri e la scrittura di 1 registro. Il clock viene messo in AND con un segnale di controllo Write e trasmesso a tutti i registri. Il segnale determina se in corrispondenza della discesa del clock il valore in input debba essere memorizzato o meno. È composto da

- 3 ingressi a 5 bit che rappresentano i due registri da leggere (rs,rt) e quello da scrivere (rd)
- Un ingresso a 32 bit che rappresenta il dato da andare a scrivere nel registro
- 2 uscite a 32 bit che rappresentano il contenuto dei registri letti

Il Register file fornisce sempre in output una coppia di registri, non significativi, se i controlli di lettura non lo sono, in tal caso, i circuiti che potenzialmente potrebbero usarli, devono ignorali.

4.2 Descrivere come avvengono le operazioni di lettura e scrittura nel Register File

Nel register file tutti i registri (32) sono collegati a due multiplexer 32:1 che ricevono come segnale di controllo i due ingressi a 5 bit che rappresentano i due registri dei quali si vuole ottenere il contenuto. A questo punto i 5 bit del primo registro vengono usati come input del primo MUX e i 5 bit del secondo registro come input del secondo MUX. Il register file fornisce sempre in output una coppia di registri, ma che non sono significativi se i controlli a 5 bit del MUX non lo sono. In questo caso i circuiti che potenzialmente potrebbero usarli devono ignorarli.

Per la scrittura invece tutti i registri sono in AND con un segnale di Write (a sua volta in AND con il clock) e con l'uscita di un decoder che decodifica il segnale di controllo in ingresso a 5 bit WriteReg#. Nel caso il clock o il Write non siano affermati, i possibili valori spuri non vengono memorizzati.

4.3 SRAM - Static RAM

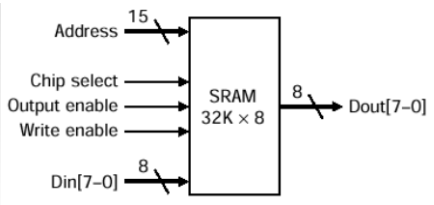
La SRAM è più veloce della memoria principale, infatti per la sua realizzazione vengono utilizzati dei latch. E' utilizzata per le memorie veloci, come le cache (tempi di accesso $0.5 - 2.5$ ns).

E' realizzata come matrice di latch $H \times W$ dove

- W è l'ampiezza, ovvero il numero di latch per ogni cella
- H è l'altezza, ovvero il numero di celle indirizzabili

Per ragioni costruttive W è spesso piccolo, e non è possibile leggere e scrivere contemporaneamente.

Esempio di chip $32K \times 8$, 32K celle da 8 bit = 256Kb

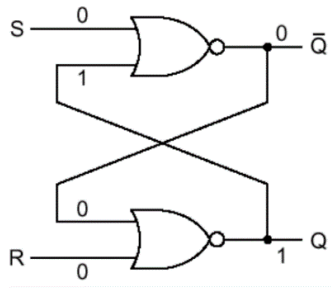


5 Circuiti sequenziali

5.1 S-R Latch

L'S-R Latch è un circuito, composto da due porte NOR concatenate, che costituisce elemento base per costruire elementi di memoria più complessi come i flip-flop. S equivale a SET, mentre R equivale a RESET. L'input del circuito sono dunque 2 bit, S e R:

- S=0, R=0: combinazione di riposo, il bit memorizzato non viene modificato
- S=1,R=0: combinazione di set, viene memorizzato il valore 1
- S=0,R=1: combinazione di reset, viene memorizzato il valore 0
- S=1,R=1: combinazione che non deve mai essere presentata al circuito, in quanto accadrebbe che Q = ¬Q = 0

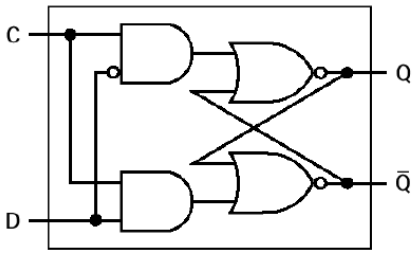


A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

5.2 Latch clockato (D-latch)

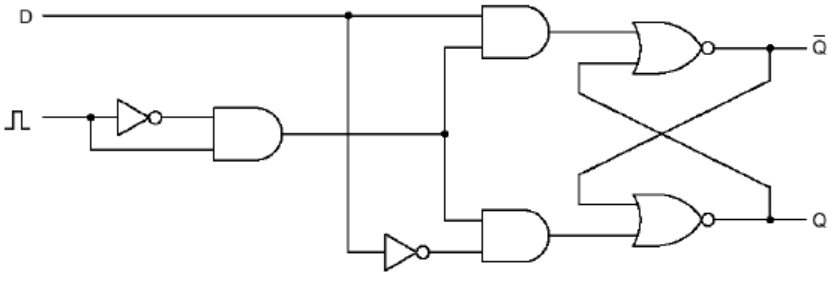
Ha lo stesso funzionamento del S-R Latch, solo che con l'aggiunta delle porte AND a monte non può mai verificarsi la condizione S=R=1 in input.

Nota che durante il periodo alto del clock il latch non esercita alcuna azione di memorizzazione. Il valore D non è stabilizzato durante il livello alto del clock, infatti può variare più volte fino alla stabilizzazione (periodo basso). Questo fenomeno è detto trasparenza del Latch.



5.3 Flip-flop semplice

E' un elemento di memoria che memorizza pressoché istantaneamente un valore D fornitogli in ingresso in corrispondenza del fronte di salita del clock (dell'impulso generato dalla salita). È un elemento edge-triggered di tipo rising triggered. Il generatore di impulsi che utilizza il flip-flop sfrutta il ritardo di propagazione delle porte per generare un impulso brevissimo in corrispondenza del fronte di salita del clock.



6 Prestazioni di un sistema

6.1 Formule per la misurazione delle prestazioni

- Il tempo di esecuzione di un programma è definito come

$$\begin{aligned} \mathrm{T}_{\mathrm{exe}} &= \#\text{ cicli} \cdot T \\ &= \frac{\#\text{ cicli}}{F} \\ &= IC \cdot CPI \cdot T \\ &= \frac{IC \cdot CPI}{F} \end{aligned}$$

- Performance:

$$\mathrm{perf}_A = \frac{1}{\mathrm{T}_{\mathrm{exe}}}$$

- Il numero di cicli:

$$\#\text{ cicli} = IC \cdot CPI$$

- MIPS, milioni di istruzioni per secondo:

$$\begin{aligned} \mathrm{MIPS} &= \frac{IC}{\mathrm{T}_{\mathrm{exe}} \cdot 10^6} \\ &= \frac{IC}{IC \cdot \frac{CPI}{F} \cdot 10^6} \\ &= \frac{F}{CPI \cdot 10^6} \end{aligned}$$

- SpeedUp, indica quante volte la macchina A è più veloce della macchina B:

$$\begin{aligned} \mathrm{speedUp} &= \frac{\mathrm{T}_{\mathrm{exe}A}}{\mathrm{T}_{\mathrm{exe}B}} \\ &= \frac{\mathrm{perf}_A}{\mathrm{perf}_B} \end{aligned}$$

6.2 Cos'è, e a cosa serve la Legge di Amdahl? Scrivere la formula e descrivere il significato delle varie componenti

La legge di Amdahl fissa un limite agli incrementi di prestazioni ottenibili (Speedup) quando introduciamo delle ottimizzazioni.

La legge di Amdahl è definita come

$$T_{\mathrm{ott}} = \frac{1}{s} \cdot T_{\mathrm{exe}} + \frac{\left(1 - \frac{1}{s}\right) \cdot T_{\mathrm{exe}}}{n}$$

dove

- $\frac{1}{s} \cdot \mathrm{T}_{\mathrm{exe}}$ è la frazione del tempo di esecuzione non modificata dalle ottimizzazioni
- $\left(1 - \frac{1}{s}\right) \cdot \mathrm{T}_{\mathrm{exe}}$ è la frazione del tempo di esecuzione modificata dalle ottimizzazioni
- n è il fattore del miglioramento ottenuto tramite le ottimizzazioni