Primo compitino Architettura II - Anno 2022/23, Versione A

Sezione Filtro

1) non si può determinare, risultato della or sul contenuto dei due registri **(D)**
2) 0x44C **(C)**
3) 16, costante da sommare **(C)**
4) Non si può sapere **(D)** (risultato propagato dalla and, hazard) / contenuto del registro $s5 **(B)**
5) contenuto del registro $t2 **(D)**
6) Contenuto del registro $t1 **(B)**
7) il flush di una istruzione in una pipeline si verifica quando una istruzione viene eliminata dalla pipeline **(C)**
8) Una pipeline aumenta la velocità di esecuzione di un fattore pari al numero di stadi **(B)**
9) Che cosa è un interrupt? Un evento non previsto che deve essere gestito dalla CPU e dal sistema operativo **(E)**
10) Le CPU multiple issue sono CPU che eseguono in parallelo più istruzioni in pipeline **(C)**

Sezione Due

1) Data la CPU N.2, quando è in esecuzione il seguente segmento di codice, sottolineare quali linee trasportano segnali utili. Identificare tutti gli hazard.

```
0x00000400  and     {$s5}, $t2, $t1
0x00000404  lw      {$s1}, 8($s0)
0x00000408  or      $t4, {$s5}, {$s1}
0x0000040C  addi    {$t1}, {$s1}, 100
0x00000410  sw      $s2, 32({$t1})
0x00000414  sub     $s2, $s0, $s2
```

In questo segmento di codice sono presenti i seguenti hazard: l'istruzione or ha due dipendenze: dalla istruzione and detiene il registro $s5 e dalla istruzione lw detiene il registro $s1. inoltre anche la addi ha una dipendenza dalla lw per via del registro $s1 e la sw ha una dipendenza dall'istruzione addi per via del registro $t1.

Quando questo segmento è in esecuzione sulla CPU in figura 2, in essa avremo caricare le seguenti istruzioni: - AND, in fase di writeback - LW, in fase di memoria - OR, in fase di execute - ADDI, in fase di decode - SW, in fase di fetch - l'istruzuone di SUB non è ancora dentro la nostra CPU.

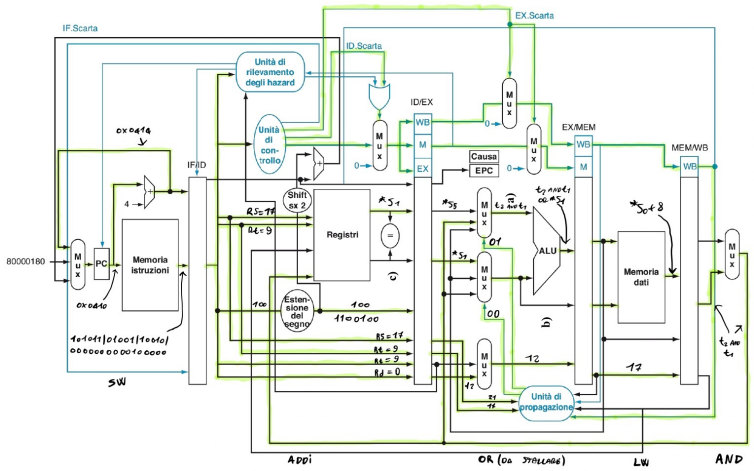

Figure 1: Percorsi segnali utili

Le linee che trasportano segnali utili sono le seguenti, analizziamole divise per fasi:

- Istruzione AND, Fase di WriteBack
  - MEM/WB.data = t2 and t1
  - MEM/WB.RegScrittura = $s5 = 21
  - MEM/WB.memtoreg = 0
  - MEM/WB.regwrite = 1
- Istruzione LW, Fase di Memory
  - EX/MEM.data = *S0 + 8
  - EX/MEM.*rt* = s1
  - EX/MEM.RegScrittura = $s1 = 17
  - MDR = MEM[*s0 + 8]
  - EX/MEM.memread = 1
  - EX/MEM.memwrite = 0
  - EX/MEM.branch = 0
  - EX/MEM.regwrite = 1
  - EX/MEM.memtoreg = 1
- Istruzione OR, Fase di EXE
  - ID/EX.*rs* = s5
  - ID/EX.*rt* = s1
  - ID/EX.rs = $s5 = 21
  - ID/EX.rs = $s1 = 17
  - ID/EX.rd = $t4 = 12
  - Operando1 = t2 and t1 (propagato)
  - Operando2 = *s1
  - ID/EX.aludataout = (t2 and t1) or *s1
  - ID/EX.alusrc1 = 01, ID/EX.alusrc2 = 00
  - ID/EX.memtoreg = 0
  - ID/EX.regwrite = 1
  - ID/EX.memread = 0
  - ID/EX.memwrite = 0
  - ID/EX.branch = 0
  - ALUop = 10
  - RegDst = 1
- Istruzione ADDI, Fase di decodifica
  - IF/ID.rs = $s1 = 17
  - IF/ID.rt = $t1 = 9
  - Operando1 = *s1
  - Operando2 = *t1
  - AluSrc1 = 00, AluSrc2 = 00
  - AluOP = 10
  - RedDst = 0
  - MemRead = 0
  - MemWrite = 0
  - Branch = 0
  - Jump = 0
  - RegWrite = 1
  - Memtoreg = 0
- Istruzione SW, fase di fetch
  - 001000|10001|01001|0000000001100100 (OPCODE|17|9|100)

In questo caso le istruzioni vengono caricare tutte nella nostra CPU pipeline, per gestire gli hazard dati dell'istruzione OR, al primo ciclo la istruzione AND propaga il suo risultato al primo operando della ALU. Al ciclo successivo per gestire l'hazard della LW eseguiamo uno stallo sulla OR, lasciandola nello stadio di EXE, cosi facendo la LW può propagare il suo dato letto in memoria al secondo operando della OR. In questo

modo la OR avrà i due operandi corretti. Tieniamo presente che andrà gestito anche hazard della ADDI e della SW.

2) Cosa sono gli interrupt e le eccezioni? Come vengono gestiti dalle architetture Intel e dalle architetture MIPS/ARM? Specificare gli elementi della CPU MIPS che sono dedicati alla gestione delle eccezioni e cosa contengono. Spiegare come la CPU in figura 2 gestisce una eccezione di Overflow (aggiungere eventuali cammini mancanti). Cosa si intende per mascheramento degli interrupt? Viene praticato nei MIPS? Come vengono gestite le eccezioni e gli interrupt dai sistemi operativi sul MIPS? Scrivere uno scheletro di funzione assembler per gestire un'eccezione.

Le eccezioni sono degli eventi non previsti che interrompono immediatamente il flusso di esecuzione di un'istruzione, le eccezioni sono generalmente problematiche interne al processore, ma possono anche essere esterne, come per esempio una situazione di overflow o l'uso di una istruzione non definita. Gli interrupt invece sono delle particolari eccezioni che fanno riferimento solamente ad eventi che hanno cause esterne al processore, per esempio la richiesta di attenzione da parte di un dispositivo I/O o un malfunzionamento hardware.

Nelle architetture Intel gli interrupt e le eccezioni vengono gestiti tramite un meccanismo detto Interrupt Vector Table, abbreviato IVT, che contiene gli indirizzi degli handler specifici per la gestione di ogni tipo di interrupt/eccezzione. Per esempio quando si verifica un interrupt o eccezione la CPU procede a consultare la IVT per identificare l'handler corretto per la sua gestione, una volta trovato la CPU salta all'indirizzo specificato dalla IVT per andare ad eseguire il codice di gestione appropriato. Esistono due tipologie di IVT:

- La intel real mode, la versione meno recente, che possiede una IVT di grandezza pari ad 1Kbyte ed è in grado di gestire fino a 256 interrupt, i primi 32 dell'elenco sono riservati alla gestione delle eccezioni, i restanti per gli interrupt. Ogni interrupt/eccezione sono associati a un gruppo di 4 byte.
- La protected mode è la versione più recente, essa ha una IVT di grandezza pari ad 2Kbyte ma gestisce lo stesso numero di interrupt della real mode in quanto ogni interrupt/eccezione è associato a un gruppo di 8 byte.

Nelle architetture MIPS/ARM gli interrupt e le eccezioni vengono gestite in modo diverso, in queste CPU è presente un register file secondario, detto coprocessore 0, nel quale sono presenti le varie informazioni sullo stato della macchina, per la gestione degli interrupt/eccezioni ci interessano i registri di stato, causa e l'exception program counter, detto EPC. Il primo step nella gestione di una interrupt/eccezione è l'identificazione della tipologia di eccezione/interrupt che si è verificato, successivamente a questo viene effettuato il salvataggio dello stato, cioè viene salvato nel registro EPC l'indirizzo dell'istruzione che ha causato l'eccezione/interrupt e nel mentre verrà anche salvato il codice della causa nel registro causa. Tramite l'operazione di flush viene effettuata l'eliminazione delle istruzioni successive che sono gia in esecuzione in quanto è probabile che queste vengano gestite in modo errato, successivamente verrà impostato il PC per poter accedere ad un indirizzo preciso che contiene l'istruzione in grado di rispondere all'eccezione riscontrata. Una volta che l'eccezione è stata risolta tramite l'indirizzo salvato all'interno del EPC si ritorna ad eseguire l'istruzione che ha causato l'eccezione oppure quella successiva, a seconda delle casistiche. Gli elementi utilizzati da una CPU MIPS per la gestione delle eccezioni sono:

- L'exception program counter, EPC, è un registro dedicato a memorizzare l'indirizzo dell'istruzione coinvolta nella eccezione/interrupt.
- Il registro causa è un registro dedicato a memorizzare la causa dell'eccezione, per esempio il codice 13 è associato all'eccezione di istruzione non definita e il codice 12 all'overflow aritmetico.
- Il registro di stato gestisce la maschera delle interruzioni e lo stato dei diversi livelli di priorità.

Da notare che le eccezioni sono problematiche più importanti da gestire rispetto agli interrupt, infatti queste ultime hanno precedenza sugli interrupt.

La CPU in figura 2 per gestire un'eccezione di overflow ha bisogno dell'aggiunta di un cammino verso l'unità di controllo dalla ALU principale, che identifica se si è verificato overflow, nel passaggio successivo se si è verificato overflow l'UC imposta ad uno i segnali di CausaWrite e EPCWrite per poter scrivere nei due registri del coprocessore zero, possiamo usare il bit di identificazione dell'overflow come bit di attivazione per i segnali di CausaWrite ed EPCwrite, in questo momento la CPU salverà nel registro EPC l'indirizzo

dell'istruzione che ha generato overflow e nel registro causa andrà a salvare il codice dell'eccezione di overflow, pari a 12. È necessario aggiungere anche un nuovo bus di controllo che andrà a gestire il multiplexer del PC per poter andare a trasferire il controllo all'indirizzo 0x80000180 che conterrà l'istruzione per rispondere all'eccezione, anche dette Exception handler. Inoltre quando si verifica una situazione di overflow nella fase di calcolo è necessario effettuare il flush delle istruzioni successive e della istruzione add coinvolta, in quanto potrebbero essere state caricate in modo errato. Una volta risolta l'eccezione procediamo a partire dall'istruzione successiva a quella che ha causato l'overflow, recuperando l'indirizzo dall'EPC.

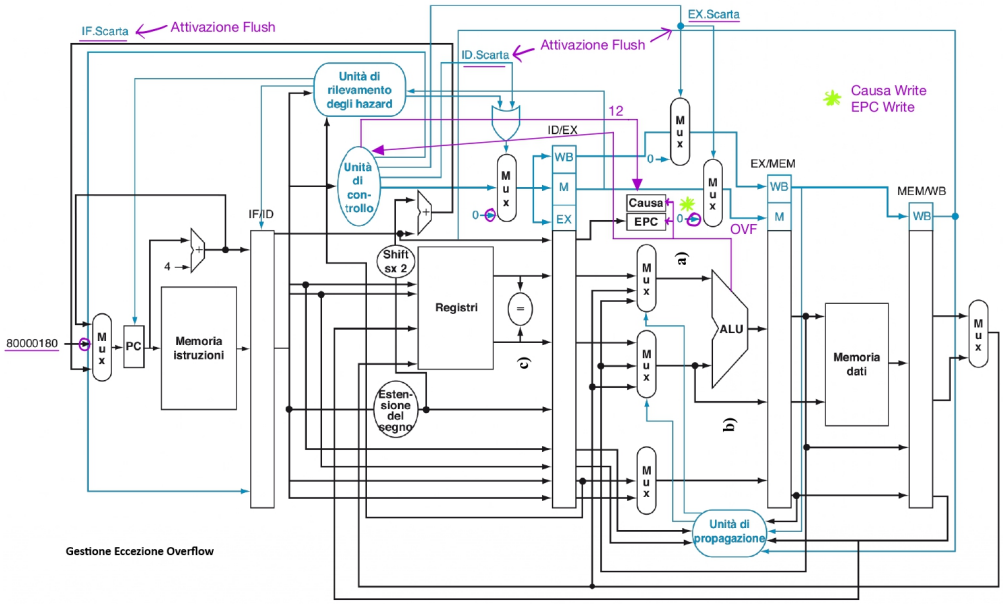

Figure 2: Gestione Eccezione Overflow

Per mascheramento degli interrupt viene intesa una sequenza di bit, nella quale ad ogni bit corrisponde un livello di interrupt, gli interrupt di un certo livello possono essere serviti solo se il corrispondente bit della maschera vale 1. Il mascheramento degli interrupt viene praticato nelle architetture MIPS ed è gestito principalmente attraverso il registro di stato e il registro di maschera, questi registri sono manipolati dal sistema operativo o kernel che gestiscono l'abilitazione degli interrupt in base alle esigenze e priorità.

Dal sistema operativo gli interrupt e le eccezioni sono gestiti nel seguente modo: Per primo passo è necessario salvare lo stato, cioè i registri importanti e l'indirizzo del PC, successivamente si risponde all'eccezione per la sua gestione, una volta risolta si procede a ripristinare lo stato precedente.

Vediamo un esempio di codice Assembler per la gestione di una eccezione:

Salvataggio dello stato nella parte riservata al SO, non in stack in quanto può far parte dell'eccezione

```
.ktext 0x80000180
sw $a0, save0
sw $a1, save1

.kdata
save0: word 0
save1: word 1
```

Risposta all'eccezione:

lettura dei registri causa e EPC:

```
mfc0 $k0, $s13 --> legge la causa dal coprocessore zero
```

```
mfc0 $k1, $s14 --> legge l'EPC dal coprocessore zero
```

estrazione della causa:

```
srl $a0, $k0, 2 --> allinea l'exception code di causa a 0
andi $a0, $a0, 0x1f --> estrae i 5 bit meno significativi dai bit di a0, a0 conterrà la causa
```

Verifica se l'eccezione è un interrupt, causa = 0, nel caso termino

```
beq $a0, $zero, dopo --> se la causa = 0, salta all'etichetta dopo
```

Stampo la causa dell'eccezione e l'indirizzo:

```
move $a1, $k1 --> sposto k1 in a1 per la visualizzazione
jal print_exception --> causa in a0, epc in a1, effettuo una chiamata di sistema per stampare
```

Ripristino dello stato:

Caricare l'indirizzo di ritorno (EPC - 4) se l'istruzione deve essere rieseguita:

```
done: --> etichetta che indica la risoluzione dell'eccezione
addi $k1, $k1, -4 --> calcoliamo EPC (k1), PC - 4
mtc0 $k1, $s14 --> portiamo il registro k1 nell'EPC
```

Azzeriamo il registro causa, k0:

```
addi $k0, $k0, 0xffff ffe0--> azzeriamo i 5 LSB
sll  $k0, $a0, 2 --> allineiamo nuovamente i bit di causa
mtc0 $k0, $s13 --> aggiorniamo il registro causa
```

Cancelliamo il flag di eccezione dal registro status e abilitiamo nuovamente gli interrupt:

```
mfc0 $k0, $s12 --> leggiamo il registro di stato dal coprocessore zero
andi $k0, 0xfffd --> impostiamo a zero il bit di eccezione
ori  $k0, $k0, 1 --> impostiamo ad uno il bit di abilitazione agli interrupt
mtc0 $k0, $s12 --> reimpostiamo il registro di stato del coprocessore zero
```

Ripristinare i registri a0 e a1 utilizzati in jal:

```
lw $a0, save0
lw $a1, save1
```

Ritorno al punto dell'interruzione:

```
eret --> porta l'indirizzo EPC nel PC
```

3) Modificare la pipeline in figura 1 perchè diventi una pipeline superscalare. Spiegare la ragione e lo scopo di tutte le modifiche più rilevanti da appartore ai diversi stadi. Che differenza c'è tra pipeline super-scalare e pipeline dotata di VLIW? Quali sono i vantaggi e gli svantaggi di un approccio rispetto all'altro. Qual'è il migliore e perchè? Descrivere come funzionano le seguenti tecniche e dire se sono tecniche principalmente software o hardware e perchè. In alcuni casi la risposta corretta può essere entrambi gli approcci. Identificare quali sono i punti forti ed i punti deboli.

Una pipeline superscalare è divisa in tre macrofasi: fetch & decode, esecuzione e commit. La prima fase si occupa di eseguire le operazioni di fetch e decodifica delle istruzioni, una volta decodificate lo scheduler si occupa della gestione delle istruzioni, andando a risolvere le eventuali criticità tramite le operazioni di ridenominazone dei registri e loop unrolling e crea dinamicamente gli issue packet che successivamente verrano inviati alle reservation station. Nel nostro caso il registro ID/EX diventa la reservation station infatti verrà aumentato di ampiezza per permettere la coda degli issue, poco prima della reservation station andiamo ad aggiungere lo scheduler, che si occupa di riorganizzare le istruzioni prima di portarle nella reservation station, a questo punto gli issue packet vengono eseguiti dalle unita funzionali dedicate per poi passare alla fase della commit unit, nel nostro disegno la commit unit, insieme al reorder buffer, li possiamo vedere come il registro MEM/WB, come collocazione temporale, infatti questi due componenti si occupano di ricevere le

istruzioni processate e tramite il reorder buffer vengono riordinate nell'ordine corretto, in quanto una CPU superscalare esegue le istruzioni fuori ordine, e se necessario la commit unit gestisce anche i feed-forwarding paths collegati con la reservation station.

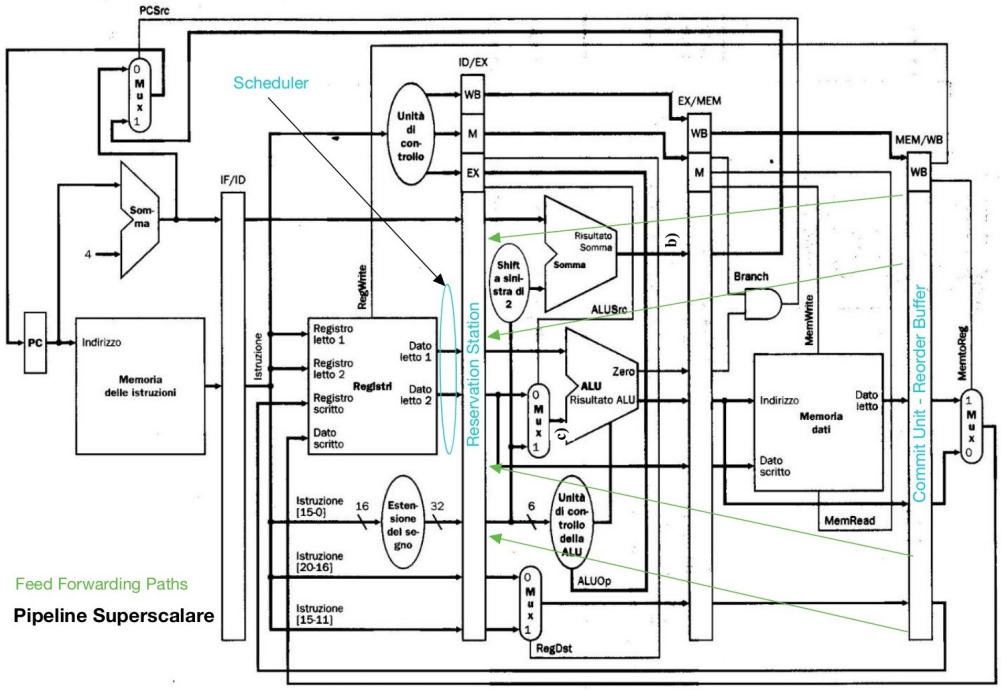

Figure 3: Pipeline Superscalare

La differenza tra una pipeline superscalare e una pipeline VLIW consiste nel fatto che la prima utilizza tecniche dinamiche per l'organizzazione degli issue packet, grazie al supporto hardware, in modo da eseguire le istruzioni in parallelo su più cammini di esecuzione. Mentre una CPU VLIW organizza le istruzioni in pacchetti di lunghe parole istruzione, detti very long instruction word, che contengono più istruzioni da eseguire contemporaneamente sui diversi cammini di esecuzione, questa organizzazione avviene in modo statico durante la fase di compilazione.

I vantaggi delle CPU superscalari sono una esecuzione dinamica delle istruzioni in modo indipendente e parallelo, sfruttando completamente le risorse hardware disponibili e una maggiore adattabilità ai carichi di lavoro, questo implica la capacità di garantire la correttezza del codice purche l'ISA sia gestita dalla CPU.

I vantaggi delle CPU VLIW sono una maggiore possibilità di ottimizzazione statica del codice, posseggono una struttura hardware più semplice, ciò implica una maggiore velocità ad eseguire certe tipologie di istruzioni sulla quale la CPU è studiata.

Gli svantaggi delle CPU superscalari sono dovuti dalla maggiore complessità hardware di quest'ultima, che ne deriva un costo energetico e progettuale maggiore.

Gli svantaggi delle CPU VLIW sono legate al compilatore, il quale se non è in grado di ottimizzare il codice per la conformazione della CPU questo può portare ad una difficoltà nella gestione delle dipendenze e ad un utilizzo inefficiente delle risorse.

Non possiamo definire quale delle due sia la migliore, in quanto dipende dal contesto nel quale vengono utilizzate, in quanto entrambe le CPU hanno i loro vantaggi e svantaggi, per esempio se il codice è molto vario e non ottimizzabile alla perfezione sono preferibili le CPU superscalari, in caso contrario le CPU VLIW sono più efficienti e meno energivore rispetto alle superscalari.

- **Predizione dei salti**: è una tecnica hardware e software per migliorare le prestazioni dei processori. Essa consiste nel predire il risultato della istruzione di salto, quindi in caso di previsione di salto effettuato subito dopo la branch si iniziano ad eseguire le istruzioni a cui punta il salto, mentre se la previsione è di salto non effettuato si procede alla normale esecuzione delle istruzioni successive alla branch. In entrambi i casi in caso di predizione errata sarà necessario un flush delle istruzioni in esecuzione, per poter riprendere la corretta esecuzione. Il punto forte di questa tecnica è il grosso risparmio di tempo nel caso la predizione fosse corretta, mentre il punto debole è che nel caso la predizione non fosse corretta avremo un maggiore costo a livello di tempo in quanto bisogna effettuare il flush delle istruzioni in esecuzione.
- **Branch prediction buffer**: è una tecnica hardware per la corretta gestione della speculazione e della predizione dei salti condizionati. Esso è una memoria cache della CPU dove vengono salvati gli esiti delle branch passate, in modo da poterli confrontare con le branch future per avere una maggiore precisione con la loro predizione di salto. I punti forti sono la riduzione degli stalli/flush e il miglioramento delle prestazioni della CPU grazie a predizioni più precise. Il punto debole è che il buffer ha una dimensione limitata quindi ha uno storico di branch limitato, inoltre all'avvio della macchina sarà vuoto, quindi c'è più probabilità che le predizioni siano sbagliate causando una perdita di tempo, la stessa probabilità la si ha in caso di istruzioni branch poco frequenti.
- **Speculazione**: è una tecnica sia hardware che software, la speculazione viene effettuata dal compilatore nel caso di CPU multiple-issue statiche mentre viene effettuata dal processore in caso di CPU multiple-issue dinamiche. La speculazione consiste nell'eseguire le istruzioni in modo speculativo anche se non si sa con certezza che vengano eseguite. Solitamente si anticipa l'esecuzione di alcune istruzioni che quasi sicuramente verranno eseguite in base a flussi di istruzioni passati, come per esempio cercare di indovinare il risultato di un salto condizionato. I punti forti di questa tecnica, in caso di speculazione corretta, sono un aumento significativo delle prestazioni, la riduzione. dei tempi di inattività delle CPU e un miglior sfruttamento delle unità di esecuzione. Il punto debole è che se la speculazione si rivela errata è stato svolto lavoro inutile e bisogna eseguire il flush di istruzioni e ripristinare la normale esecuzione delle istruzioni, causando una perdita di tempo.
- **Parallelizzazione dell'esecuzione**: è una tecnica sia software che hardware, in quanto è necessaria una CPU in grado di gestire l'esecuzione parallela delle istruzioni, quindi una CPU dotata di più unità funzionali, ma la parallelizzazione è gestita anche dal compilatore che si occupa di riordinare le istruzioni in modo corretto per massimizzare l'esecuzione parallela ed evitare che si creino criticità. Il punto forte di questa tecnica è l'aumento della velocità di esecuzione delle istruzioni a seconda della quantità di cammini disponibili. Il punto debole è la necessità di ottimizzare le istruzioni per far si che vengano eseguite nel miglior modo ed evitando che si generino criticità che possono causare un elevato bisogno di tempo per la loro gestione.
- **Pipeline superscalari**: sono una tecnica prettamente hardware, in quanto sono una tipologia di CPU nelle quali le istruzioni vengono processate in sequenza ed il processore si occupa delle gestione delle criticità e della loro corretta esecuzioni in modo parallelo tramite i cammini di esecuzione disponibili. I punti forti delle pipeline superscalari sono un notevole incremento delle prestazioni dovuto dalla parallelizzazione e dai diversi cammini di esecuzione, la scalabilità della CPU cioè l'implementazione di maggiori unità funzionali per la gestione di carichi più elevati. I punti deboli sono la maggiore complessità a livello circuitale, sopratutto se si necessità di una grossa scalabilità, un maggiore rischio di hazard dovuto dall'esecuzione parallela delle istruzioni se non correttamente implementate con il software adeguato.
- **Esecuzione fuori ordine**: è una tecnica hardware, adottata principalmente dalle CPU a pipeline superscalari. In quanto queste ultime sono in grado di eseguire le istruzioni fuori ordine e grazie alla presenza della commit unit e del reoder buffer, che una volta finita l'esecuzione delle istruzioni, si occupano di riordinare il codice nel modo corretto in modo da rendere il risultato indifferente agli occhi dell'utilizzatore. I punti forti di questa tecnica sono dati dall'utilizzo efficiente di tutte le risorse della CPU come per esempio i diversi cammini di esecuzione, una migliore gestione delle dipendenze e degli stalli. I punti deboli sono dati da una maggiore quantità di flush necessari in caso di una speculazione errata e quindi una maggiore perdita di tempo e un aumento di complessità del circuito della CPU dovuto anche alla gestione della riorganizzazione finale del codice.

- **Reservation station**: è una tecnica hardware, in quanto si tratta di un componente delle CPU a pipeline superscalari. Questi sono dei buffer, gestiti dallo scheduler, che contengono le istruzioni e gli operandi che sono in attesa che le unità funzionali della CPU siano disponibili. I punti forti sono il poter eseguire le istruzioni fuori ordine, la gestione delle dipendenze e l'utilizzo efficiente delle risorse della CPU. Il punto debole è garantire la corretta gestione delle istruzioni per evitare stalli e criticità.
- **Buffer di riordino**: è una tecnica hardware, in quanto si tratta di un componente delle CPU a pipeline superscalari. Il suo compito è quello di riordinare il risultato delle istruzioni per garantire che vengano scritti nei registri nell'ordine corretto, in quanto nelle pipeline superscalari le istruzioni vengono eseguite in modo non ordinato. I punti forti sono la possibilità di eseguire istruzioni fuori ordine in modo sicuro, con la conseguente miglior gestione delle dipendenze ed criticità tra le istruzioni. Il punto debole principale è dovuto dalla richiesta di maggiori risorse hardware.
- **Ridenominazione dei registri**: è una tecnica sia hardware che software, infatti in alcune CPU è gestita dal compilatore tramite la ridenominazione dei registri di alcune istruzioni che durante l'esecuzione avrebberò causato degli hazard, mentre nelle CPU superscalari la ridenominazione è gestita internamente dai registri locali appositi che la eseguono quando necessaria. Solitamente questa tecnica viene utilizzata per l'ottimizzazione dell'esecuzione dei cicli. I punti forti sono la eliminazione delle dipendenze dei dati, il miglioramento delle prestazioni e un aumento del livello di parallelismo. I punti deboli sono un aumento di lavoro da parte del compilatore, se utilizzato, e un aumento di occupazione di risorse dovuto dal maggior utilizzo di registri.
- **Issue**: è una tecnica hardware, gli issue sono l'insieme dei segnali utili al fine dell'esecuzione di un'istruzione, cioè il numero dei registri, l'opcode, i segnali di controllo ecc…, in altre parole l'issue è l'insieme di segnali che vengono ottenuti a partire da un'istruzione dopo che è stata digerità nella fase di decodifica.

4) Come si implementa l'esecuzione vettoriale in una pipeline super-scalare? Mostrarlo modificando un cammino di esecuzione di una pipeline non dotata di esecuzione vettoriale.

Per implementare l'esecuzione vettoriale in una pipeline super scalare è necessario raggruppare le istruzioni della stessa tipologia in vettori, in modo che la nostra CPU li esegua in massa portandosi a dietro solamente una volta i segnali di controllo, in quanto l'intero vettore conterrà solamente istruzioni della stessa tipologia. Questa CPU sarà dotata di una ALU maggiorata in quanto le CPU vettoriali richiedono di eseguire la stessa operazione su elementi adiacenti, questo implica una struttura modulare, infatti dobbiamo avere all'interno della nostra CPU una ALU che ci permetterà di eseguire una somma a 32 bit, due somme a 16 bit, quattro somme diverse a 8 bit, se necessario, andando ad eseguire tagli sulla catena dei riporti sotto il controllo della reservation station.

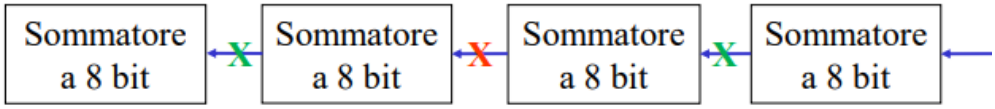

Figure 4: Modifica vettoriale

5) Spiegare chiaramente cosa si intende per stallo e cosa si intende per flush di una pipeline.

Per stallo viene inteso quando in una CPU una istruzione viene fermata nel tempo in attesa della risoluzione delle criticità che si sono verificate, quindi se durante una esecuzione di una sequenza di istruzioni alcune di queste causano criticità è necessario eseguire uno stallo, cioè il blocco di esecuzione della istruzione attuale, la quale presenta criticità, andando a inserire delle istruzioni nop, not operation, per far si che nel frattempo la criticità venga risolta e l'istruzione attuale possa essere eseguita correttamente. Le istruzioni nop vengono create andando ad impostare a zero tutti i segnali di controllo, in questo modo si hanno delle istruzioni depotenziate che durante la loro esecuzione non avranno effetti e ci permetto di guadagnare tempo per far si che si risolva la criticità. Per esempio, avendo le seguenti istruzioni: 0x04 lw $s0, 16(s1)$ e 0x08 add $t0, $s0, $s1, l'istruzione di add ha una dipendenza dalla istruzione di LW, quindi dobbiamo attendere che quest'ultima sia in fase di WriteBack per poter effettuare la corretta propagazione del dato all'istruzione add, per cui è necessario andare ad inserire una nop tra le due istruzioni per avere una corretta esecuzione

della add.

Per flush di una pipeline si intende la cancellazione di una o più istruzioni all'interno della pipeline perchè non più valide, solitamente un flush è necessario quando si verificano dei branch mal predetti e quindi sono stati eseguite alcuni istruzioni errate oppure quando si verificano delle eccezioni, dove la CPU viene svuotata dalle istruzioni successive a quella che ha causato l'eccezione per poter gestire correttamente quest'ultima. Per esempio se l'istruzione su cui è necessario effettuare il flush si trova nella fase di fetch per eseguire il flush indichiamo all'hazard detection unit di effettuare l'operazione di reset del registro IF/ID cosi otteniamo una istruzione contenente soli zero, che equivale ad una SLL $zero, $zero 0 che se eseguità non avrà alcun effetto, se ci troviamo nelle altre fasi l'operazione di flush può essere implementata tramite l'inserimento di una NOP.

6) Riscrivere il codice seguente in modo che sia eseguito nel mior tempo possibile su una pipeline con 4 cammini di esecuzione di cui 3 general purpose e 1 dedicato solo alle istruzioni di memoria. Qual'è lo speed-up? Applicare lo srotolamento dei cicli per un numero massimo di 8 iterazioni del ciclo. Si supponga di avere a disposizione un numero di registri interni di pipeline sufficientemente grande e che si possa applicare la ridenominazione dei registri.

```
Ciclo:  lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        addi    $s1, $s1, -4
        bne     $s1, $zero, Ciclo
        or      $s6, $s7, $s5
```

GenP	GenP	GenP	MEM
nop	nop	nop	LW
nop	nop	nop	nop
ADDU	nop	nop	nop
ADDI	nop	nop	SW
nop	nop	nop	nop
BNE	nop	nop	nop

le istruzioni per ciclo, IPC, di questo codice è pari ad 0,8.

Andando ad riorganizzare il codice in modo da gestire meglio le dipendenze, possiamo spostare la addi come seconda istruzione del ciclo data la sua quasi totale indipendenza, per via dello spostamento dobbiamo aggiungere 4 davanti al registro target della SW, arrivando al seguente codice ottimizzato:

```
Ciclo:  lw      $t0, 0($s1)
        addi    $s1, $s1, -4
        addu    $t0, $t0, $s2
        sw      $t0, 4($s1)
        bne     $s1, $zero, Ciclo
        or      $s6, $s7, $s5
```

GenP	GenP	GenP	MEM
nop	nop	nop	LW
ADDI	nop	nop	nop
ADDU	nop	nop	SW
BNE	nop	nop	nop

le istruzioni per ciclo, IPC, di questo codice riordinato è pari ad 1.25, in questo modo abbiamo migliorato le prestazioni, raggiungendo uno speed-up pari ad 1.5, rispetto al codice iniziale.

Possiamo eseguire una ulteriore ottimizzazione, andando ad applicare lo srotolamento dei cicli per un massimo di 8 iterazioni del ciclo. presumendo di avere a disposizione un numero elevato di registri possiamo effettuare anche la ridenominazione dei registri, il nostro ciclo diventa il seguente:

```
Ciclo:  addi    $s1, $s1, -28
        lw      $t0, 28($s1)
        lw      $t1, 24($s1)
        lw      $t2, 20($s1)
        lw      $t3, 16($s1)
        lw      $t4, 12($s1)
        lw      $t5, 8($s1)
        lw      $t6, 4($s1)
        lw      $t7, 0($s1)
        addu    $t0, $t0, $s2
        addu    $t1, $t1, $s2
        addu    $t2, $t2, $s2
        addu    $t3, $t3, $s2
        addu    $t4, $t4, $s2
        addu    $t5, $t5, $s2
        addu    $t6, $t6, $s2
        addu    $t7, $t7, $s2
        sw      $t0, 28($s1)
        sw      $t1, 24($s1)
        sw      $t2, 20($s1)
        sw      $t3, 16($s1)
        sw      $t4, 12($s1)
        sw      $t5, 8($s1)
        sw      $t6, 4($s1)
        sw      $t7, 0($s1)
        bne     $s1, $zero, Ciclo
        or      $s6, $s7, $s5
```

GenP	**GenP**	**GenP**	**MEM**
ADDI	nop	nop	nop
LW t0	LW t1	LW t2	LW t3
LW t4	LW t5	LW t6	LW t7
ADDU t0	ADDU t1	ADDU t2	SW t0
ADDU t3	ADDU t4	ADDU t5	SW t1
ADDU t6	ADDU t7	SW t2	SW t3
SW t4	SW t5	SW t6	SW t7
BNE	nop	nop	nop

le istruzioni per ciclo, IPC, ora sono 3.25, con uno speed-up pari a 4 rispetto al nostro codice iniziale.

Grazie all'implementazione del loop unrolling con la ridenominazione dei registri siamo riusciti ad eliminare tutte le dipendenze tra le nostre istruzioni e quindi siamo riusciti ad raggiungere una ottimizzazione ottimale del nostro ciclo.

7) Modificare la pipeline in figura 2, in modo che gestisca correttamente l'hazard generato da una istruzione aritmetico logica seguita da una istruzione di salto condizionato, fare un esempio.

le modifiche necessarie alla CPU in figura 2 per la gestione dell'hazard generato da un istruzione aritmetico logica seguita da una branch, come per esempio 0x400 add $t2, $s4, $s3, 0x404 beq $t2, $s6, "tag", sono le seguenti:

l'implementazione nell'hazard detection unit della seguente operazione per individuare l'hazard IF (IF/ID.opcode == "BEQ") and (ID/EXE.rd == IF/ID.rs or ID/EXE.rd == IF/ID.rt) then stallo del PC e IF/ID, per effettuare questo confronto dobbiamo aggiungere un bus per portare ID/EXE.rd all'interno della hazard detection unit, le informazioni dell'istruzione IF/ID gia le possiede.

Se il confronto dovesse risultare positivo, quindi c'è un hazard, bisogna procedere all'inserimento di una bolla nella fase di EXE, per far procedere l'istruzione ADD in fase di MEM e mantenere l'istruzione BEQ in fase di DEC. per inserire la bolla l'unità di controllo attiva il mux "ID.Scarta" per selezionare "0", in questo modo i segnali di controllo verranno azzerati.

A questo punto in fase di MEM avremo la ADD e nel bus EX/MEM.AluOut è presente il dato pronto, l'unità di propagazione procederà a propagare questo dato all'interno dei multiplexer del comparatore dell'istruzione

BEQ in fase di DEC, i multiplexer sono stati aggiunti per poter selezionare il registro corretto da comparare, quello che risulta vero tra (ID/EXE.rd == IF/ID.rs or ID/EXE.rd == IF/ID.rt) verrà sostituito con il dato propagato dalla fase di MEM. Oltre ai multiplexer è necessario aggiungere il cammino di propagazione dalla fase di MEM alla fase di DEC e due bus in uscita dall'unità di propagazione per andare a controllare i due nuovi multiplexer.

A questo punto verrà effettuato il confronto tra i due dati e si procede con la normale esecuzione delle istruzioni, data la risoluzione della criticità.

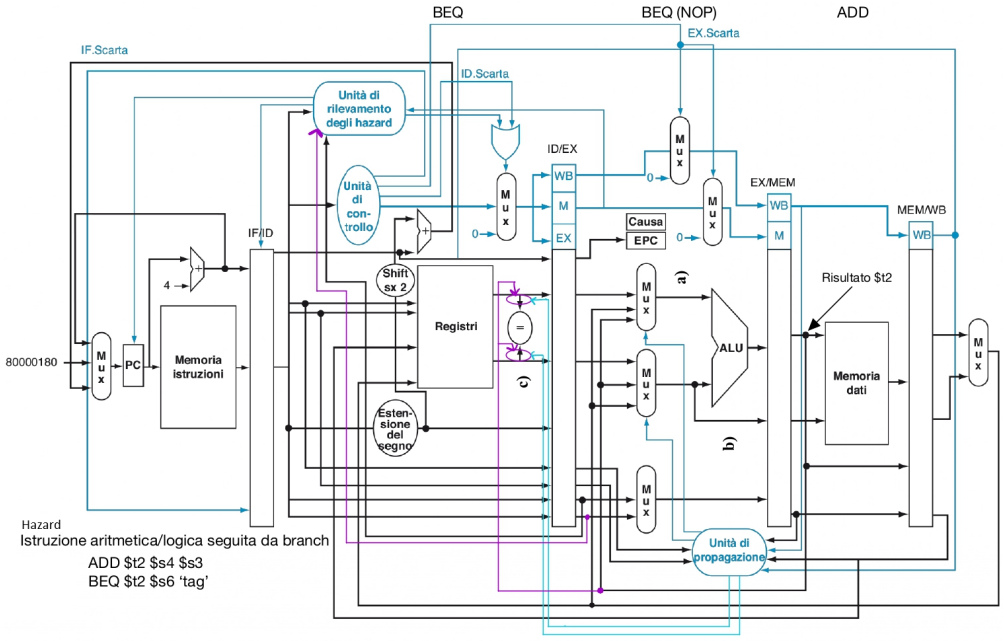

Figure 5: Pipeline Gestione Hazard ADD BEQ

Domande aggiuntive prese dai vecchi compitini, veramente vecchi

3) Descrivere come funzionano le seguenti tecniche e dire se sono tecniche principalmente software o hardware e perchè. In alcuni casi la risposta corretta può essere entrambi gli approcci. Identificare quali sono i punti forti ed i punti deboli.

- **Superpipeline**: è una tecnica prevalentemente hardware, in quanto si tratta di una tipologia di CPU, questa CPU si basa sul modello pipeline e possiede molte più fasi rispetto ad una pipeline normale, in questo modo riusciamo ad avere più istruzioni in esecuzione nella nostra CPU e un maggiore throughput. infatti il punto di forza di questa CPU è un elevato throughput rispetto alle pipeline normali. il punto debole invece è che risulterà più complessa la gestione delle dipendenze e degli hazard, oltre che ad essere più frequenti.
- **Branch delay slot**: è una tecnica software che ci dice che le istruzioni immediatamente successive ad un salto vengono sempre eseguite, sia se il salto venga preso o che non venga preso e quindi sfruttiamo questi slot per poter eseguire istruzioni indipendenti dall'istruzione di salto. il punto forte di questa tecnica è che garantisce un maggiore sfruttamento della CPU, il punto debole è che non è sempre possibile ottimizzare il codice per far che ciò avvenga.
- **Hazard**: è una tecnica hardware, rappresenta dei problemi che si possono verificare all'interno di una CPU pipeline quando l'esecuzione di una istruzione ha delle dipendenze dalla istruzione precedente che non ha ancora finito la sua esecuzione. esistono tre tipologie di hazard: hazard sui dati, hazard di controllo e hazard strutturali. il punto di forza è che la loro corretta gestione migliora le prestazioni ma di conseguenza introduce una maggiore complessità della CPU, che è un punto debole.

- **Bolla**: è una tecnica hardware, questa viene utilizzata per la correzzione degli hazard, è una istruzione vuota, anche detta NOP, e serve al nostro processore per permettere di far avanzare delle istruzioni per soddisfare delle dipendenze delle istruzioni successive. l'istruzione NOP viene creata andando ad impostare tutti i suoi segnali di controllo pari a zero, in questo modo la sua esecuzione non causerà modifiche. il punto forte è che permette una semplice soluzione degli hazard, il punto debole è che riduce il throughput e l'efficienza della CPU, in quanto ferma una istruzione e le sue successive per uno o più cicli.
- **Stallo**: è una tecnica hardware, consiste nel blocco temporaneo della pipeline per via di una istruzione che non può essere eseguita immediatamente per via di un hazard, solitamente questa tecnica viene utilizzata insieme alle istruzioni bolla o NOP. il punto forte è che evita risultati errati o incosistenti delle nostre istruzioni eseguite, ma il punto debole è una riduzione delle prestazioni della CPU.
- **Flush**: è una tecnica hardware, questa coniste nello svuotamento della pipeline da una o più istruzioni, per esempio a seguito di un errore nella predizione di un salto andando ad eliminare le istruzioni che erano state eseguite in modo speculativo, oppure in caso di una eccezione/interrupt. l'operazione di flush viene eseguita tramite il reset dei registri intermedi o impostando i segnali di controllo a zero. il punto forte è che permette di mantenere una corretta esecuzione, ma il punto debole è che riduce le prestazioni della CPU.
- **Architettura vettoriale**: è una tecnica hardware e software, questà è una archittettura in cui una istruzione opera su più dati contemporaneamente, spesso viene utilizzata per il calcolo scientifico e nelle schede video GPU. I punti di forza sono l'ottimizzazione per operazioni su grandi quantita di dati ed un elevato parallelismo. Il punto debole è che non è sempre efficiente in caso di carichi di lavoro non vettorializzabili.
- **Pipeline VLIW**: è una tecnica hardware e software, esse sono una tipologia di pipeline che utilizza istruzioni molto lunghe contenenti più operazioni da eseguire in parallelo. è anche una tecnica software in quanto la schedulazione delle istruzioni è gestita completamente dal compilatore, l'hardware non la supporta. i suoi punti di forza sono una minore complessità hardware rispetto alle pipeline superscalari e alta efficienza rispetto a programmi ben ottimizzati. I punti deboli sono dovuti dall'obbligo di un buon compilatore che organizza le istruzioni e una inefficienza in caso di dipendenze complesse.
- **Parallelizzazione a livello di parola**: è una tecnica hardware, che sfrutta la possibilita di eseguire operazioni su parole di dati più grandi. Il suo punto fore è la maggiore efficienza nell'elaborazione di dati numerici. Il punto debole è che richiede hardware compatibile.
- **Parallelismo implicito ed esplicito**: è una tecnica sia hardware che software, il parallelismo implicito è gestito autonomamente dal processore o dal compilatore, senza che il programmatore lo comunichi. Mentre il parallelismo esplicito è gestito dal programmatore che indica esplicitamente quale parti di codice possono e devono essere eseguite in parallelo. i punti forti di queste tecniche sono le maggiori prestazioni che garantiscono e il punto debole è che non sempre è possibile effettuare del parallelismo, dipende dalla CPU e il parallelismo esplicito richiede abilità da parte del programmatore.

3) spiegare come funziona l'unità di propagazione

l'unità di propagazione ci aiuta nella gestione degli hazard andando ad evitare alcuni situazioni di stallo e di conseguenza aumentare la velocità generale di esecuzione della nostra CPU pipeline. Quando si verifica un hazard, essa permette di retropropagare alcuni dati già pronti alle istruzioni che ne necessitano, per esempio se abbiamo una istruzione ADD che salva il dato nel registro $t0 ma l'istruzione successiva anche lei utilizza $t0, quest'ultimo non sarà aggiornato in quanto la ADD non ha ancora salvato in memoria il dato prima che l'istruzione successiva lo ha prelevato, quindi l'istruzione successiva ha un dato sbagliato del registro $t0, a questo punto interviene la nostra unità di propagazione che va a prendere il risultato della ADD, che in fase di memoria è gia presente, e lo porta all'ingresso del multiplexer della ALU dell'istruzione successiva, in questo modo abbiamo risolto la criticità senza dover eseguire uno stallo.

Utilità varie

Campo	0	rs	rt	rd	shamt	funz
Posizione dei bit	31–26	25–21	20–16	15–11	10–6	5–0

a. Istruzioni di tipo R

Campo	35 or 43	rs	rt	indirizzo
Posizione dei bit	31–26	25–21	20–16	15–0

b. Istruzioni di load/store

Campo	4	rs	rt	indirizzo
Posizione dei bit	31–26	25–21	20–16	15–0

c. Istruzioni di salto condizionato

Istruzione (OpCode)	Reg Dst	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	AluOP	Jump
R (000000)	1	0	0	1	0	0	0	10	0
lw (100011)	0	1	1	1	1	0	0	00	0
sw (101011)	x	1	x	0	0	1	0	00	0
addi (001000)	0	1	0	1	0	0	0	00	0
beq (000100)	x	0	x	0	0	0	1	01	0
j (000010)	x	x	x	0	0	0	0	xx	1

Codice operativo istruzione	ALUOp	Operazione eseguita dall'istruzione	Campo funzione	Operazione dell'ALU	Ingresso di controllo alla ALU
lw	00	load di 1 parola	XXXXXX	somma	0010
sw	00	store di 1 parola	XXXXXX	somma	0010
Branch equal	01	salto condizionato all'uguaglianza	XXXXXX	sottrazione	0110
Tipo R	10	somma	100000	somma	0010
Tipo R	10	sottrazione	100010	sottrazione	0110
Tipo R	10	AND	100100	AND	0000
Tipo R	10	OR	100101	OR	0001
Tipo R	10	set less than	101010	set less than	0111