Secondo parziale

Datapath e controllo del percorso dei dati

Se il capitolo 1 spiegava che le *performance* di un computer fossero determinate da tre fattori:

- Tempo del clock
- Numero di clock per ogni istruzione
- Numero di istruzioni

Ed il capitolo 2 spiegava che il compilatore e l'ISA determinano quante richiedeva un programma per essere eseguito, allora adesso possiamo dire che l'implementazione del processore determina entrambe le cose.

In questo capitolo si studiano i principi e le tecniche per implementare un processore.

Implementazione base del MIPS

L'Isa utilizzato determina molti degli aspetti dell'implementazione, e come la scelta di un'implementazione piuttosto che un'altra possa determinare differenti clockrate.

Per ogni istruzione, i primi due step sono sempre gli stessi

- Invio del Program Counter alla memoria che contiene il codice, e prelevare l'istruzione
- Leggo uno o due registri, utilizzando i campi dell'istruzione per selezionare i registri da leggere (dipende dall'istruzione)

Dopo questi due step le azioni richieste per completare un'istruzione dipendono dalla classe dell'istruzione. (Fortunatamente tendenzialmente sempre le stesse sono per ogni classe, perchè il mips semplifica l'implementazione facendo esecuzione di molte delle classi di istruzioni simili.)

Esempio: esclusa la jump tutte usano l'ALU dopo aver letto i registri. Le istruzioni di memory reference usano l'ALU per il calcolo dell'indirizzo, e sia per operazioni aritmetico logiche che per le operazioni da eseguire e le branch per le comparazioni.

Chiaro che dopo avere usato l'ALU per completare le varie istruzioni son diverse le operazioni da fare. Infatti la **Memory Reference** necessiterà di accedere in memoria sia per lettura che per scrittura, mentre una operazione aritmetico logica deve scrivere il risultato in memoria o su un registro.

Per la branch invece potremmo aver bisogno di cambiare il PC in base all'esito del confronto, altrimenti si incrementa di 4 il PC.

Nella figura 1: (Foglio a parte)
Si implementa una visione ad alto livello della MIPS implementation, sebbene mostri un flusso dei dati al processore, omettiamo due aspetti dell'esecuzione che sono:

1) In alcune parti il disegno mostra che i dati vanno ad una unità come se fossero da fonti diverse, ad esempio il valore scritto nel pc può venire pure da due sommatori diversi, ed il dato scritto nel Reg File può venire sia dall'ALU che magari prende dati da un registro ed il campo immediato. In pratica linee dei dati non si possono combinare, a meno che si aggiunga un elemento logico che sceglie tra le multiple risorse -> IL MULTIPLEXER.

2) E' stato omesso anche che i diversi tipi delle unità devono essere controllate in base all'istruzione, nel senso tipo che il data memory DEVE leggere sia in una lettura che in una scrittura

---

Il Percorso dei dati

Non è altro che il flusso logico che si segue per fare sì che un'istruzione venga eseguita. Nel senso, si studiano i passaggi che i dati fanno attraverso i componenti interni, in modo da poter controllare questo percorso.

Uno dei primi problemi che si introduce è quello della sovrapposizione non risolvibile con saldature, quindi si devono usare dei Mux dei nuovi registri, in modo da modificare il percorso.

Realizzazione di un datapath

- Stabiliamo un set di istruzioni da implementare
- Identifichiamo i componenti del datapath (alu, register file, ecc)
- Stabiliamo il metodo di clocking
- Assembliamo il datapath e identifichiamo i segnali di controllo
- Analizziamo l'implementazione di ogni istruzione per determinare il setting dei segnali di controllo
- Assembliamo la logica di controllo (non la logic unit, ma proprio la logica del controllo)

Esempio della fase di Fetch, Decode e Execute

Fetch (Figura 2)

- Leggo l'istruzione dalla ram e la salvo nell'istruction register
- L'indirizzo di memoria che indica l'istruzione da leggere si trova nel program counter
- Dopo la lettura dell'istruzione, si incrementa di 4 il pc per passare all'istruzione successiva (si usa l'alu)

Decode

- Decodifico i campi dell'istruzione per decidere i passi necessari per l'istruzione
- Il MIPS (processore) legge i campi dell'istruzione e
- Identifica il tipo di istruzione da eseguire (opcode e func code se necessario)

Execute

- Eseguo i passi necessari per eseguire l'istruzione

Formato R-Type Figura 3

Il register file contiene tutti i registri ed ha due porte di lettura ed una di scrittura.

Il register file spara sempre fuori il contenuto del registro corrispondente alla lettura degli input, in output, non son richiesti input di controllo.

Di contro però un register write deve essere esplicitamente indicato per assertare il segnale di controllo in scrittura. Ricordiamo sempre che le scritture sono edge triggered, quindi tutti i write inputs) che sono i valori da scrivere, i numeri di registro ed il segnale di controllo devono essere validi al clock edge

Siccome scrivere sul reg file è edge triggered, il nostro design può legalmente leggere e scrivere lo stesso registro con un solo ciclo di clock, la lettura prenderà il valore scritto in un ciclo precedente, mentre il valore scritto sarà disponibile ad una lettura in un seguente ciclo

Gli input che portano il numero di registro al register file sono tutti grossi 5 bit, invece le linee che portano i dati sono a 32 bit, le operazioni da eseguire dall'alu son controllate dall'alu op signal, che ha 4 bit di selezione.

E lo zero? Praticamente è una zero detection dell'alu che serve per implementare branch. L'overflow non sarà richiesto fino alle eccezioni.

Execute: load e Store (Figura 4)

La memory unit è un elemento di stato con:
• Input: Registro e dato da scrivere
• Output: singolo per il risultato di lettura

Ci sono dei controlli separati di lettura e scrittura malgrado solo uno di questi può essere assertato su ogni clock. La memory unit necessita un segnale di lettura, a differenza del reg file, leggere il valore di un indirizzo non valido può causare problemi.

La sign extention praticamente ha 16 bit di input che vengono estesi in un risultato a 32 bit in out.

La data memory e edge triggered per le scritture, mentre i chip standard di memoria hanno un write enable signal che praticamente si usa per le scritture. Questo non è edge triggered ma il nostro design edge triggered può essere facilmente adattato ai real memory chips

---

Execute: Beq (Figura 5)

Il datapath per una branch utilizza l'alu per valutare la condizione del branch ed un adder separato che computa il target della branch coe somma del PC incrementato ed i 16 bit (sign extended) dell'istruzione. (branch displacement), shiftato di due bit.

L'unità etichettata "shift left 2" (Ω) indirizza il segnale tra input e output che aggiunge 00~tw0 al low order end del campo sign extended offset. Non c'è bisogno di un vero e proprio hardware shift. siccome l'ammontare dello shift è costante, e siccome sappiamo che l'offset è sign extended da 16 bit, lo shift solleverà solo i bit di segno.

La logica di controllo è utilizzata per decidere se l'incremented PC o il target della branch vadano a rimpiazzre il PC basato sullo Zero dell'out della alu.

Visione astratta del datapath

~~~~~

Multiplexer per integrare i vari componenti

~~~~~

Metodologia di clocking

- Single clock
  - Ciclo singolo di lunghezza fissa uguale al tempo per eseguire l'istruzione più lunga
  - Ogni istruzione è eseguita in un ciclo di clock
  - Poco efficiente, si spreca tempo

- Le istruzioni più semplici vengono rallentate e soprattutto vengono replicate unità funzionali (memoria, alu)

- Multiciclo
  - Ciclo di lunghezza più corta
  - Ogni istruzione è costituita da più cicli
  - Istruzioni di tipo diverse valori diversi del ciclo di clock, cioè in cicli di clock differenti
  - Qualsiasi istruzione si esegue in più cicli di clock, e si hanno istruzioni di diverso tipo
  - Le unità funzionali vengono usate più volte durante l'esecuzione della stessa istreuzione in cicli differenti, meno replicazioni
  - Si usano registri aggiuntivi per memorizzare i risultati parziali nell'esecuzione delle istruzioni

---

Unendo tutte queste parti

Combinando assieme le tre parti otterremo uno schema come in figura 1, quindi in pratica abbiamo preso quello che era tutte le componenti di prima e le abbiamo unite MA sono state aggiunte chicche come lo shift prima dell'adder in alto a destra

In più si aggiunge un mux che prende in input read data 2 + il sign extended dell'istruzione da memoria, e in out esce sull'alu

Alla fine vengono aggiunti due multiplexer, che serve per integrare le branch. Ancora però NON puoi fare la jump, per ora abbiamo un datapath (Ridisegnato in figura 6) che esegue load store, operazioni alu e branch (singolo ciclo)

Osservazione:
Come è possibile notare, a gestire tutti i vali segnali di controllo ci penserà proprio la control unit, dalla quale escono una serie di fili alle varie componenti. In figura 7 è illustrata una control unit che prevede anche la presenza di un Jump

---

Implementazione della jump

Si aggiunge un multiplexer addizionale (in alto a destra rispetto alla figura 6) che praticamente sceglie tra un jump target o un branch target o semplice esecuzione sequenziale.

Questo Mux in pratica è controllato dal jump signal, e l'indirizzo target è ottenuto shiftando i 26 bit più a destra (piccini) dell'istruzione di due bit. Cioè alla fine si aggiungono due bit a 0 come bit meno significativi e poi si concatenano i 4 bit più grandi del PC+4 come bit più grandi, in modo da avere il final e effettivo registro da 32 bit.

Datapath multi-ciclo

Registri aggiuntivi:

- Ci sono dei registri aggiuntivi che servono a memorizzare valori intermedi usati in un ciclo di clock successivo per continuare l'esecuzione della stessa istruzione
  - Istruction register
  - Mdr, memory data register
  - A,B che praticamente sono registri tra reg file e ingresso dell'alu
  - Alu out, che sarebbe detto semplicemente il valore di output dell'alu

Riutilizzo delle unità funzionali

- L'alu è usatra non solo per operazioni aritmetico logiche ma anche per calcolare l'indirizzo dei salti e per incrementare il PC
- La memoria è usata sia per leggere le istruzioni che per scrivere o leggere i dati

Gli elementi chiave del multiciclo alla fine sono la memoria condivisa, e una singola alu che è condivisa tra più istruzioni E le connessioni tra queste unità condivise.

L'uso di unità funzionali condivise richiede l'aggiunta o l'allargamento dei multiplexer come registri temporanei che mantengono i dati tra cicli di clock della stessa istruzione. I registri addizionali sono degli IR (istruction), la memory register (MDR), A,B, e Alu out (quelli segnati qui sopra)

In figura 8 c'è una spiegazione più approfondita e chiara.

Passi per eseguire le istruzioni

Considerando che qualsiasi istruzione è eseguita in più passi, e che quindi ogni passo è eseguito in un ciclo di clock (corto) è necessario il bilanciamento della quantità di cose da fare per ogni passo.

Un'unità funzionale viene utilizzata solo una volta durante uno stesso ciclo, ed al termine di ogni ciclo di clock i valori intermedi vengono memoriazzati nei registri addizionali e ci rimangono per il ciclo successivo

Quali sono precisamente questi passi?

1. Fetch: incremento del program counter e fetch istruzione
2. Decode: decodifico l'istruzione e leggo registri e calcolo l'indirizzo per un branch

3. Execute: eseguo le istruzioni relative a rtype oppure calcolo di memoria oppure branch oppure jump

4. Execute: completo e termino la mia r_type oppure accedo alla memoria

5. Execute: scrivo il registro (solo per la lw)

Chiaro che l'istruzione più pesante di tutte abbia 5 passi per eseguirla mentre invece di minima son 3
E la loadword è sempre la più pesante di tutte le istruzioni

Passo 1: Fetch (Tutte le istruzioni)

Operazioni:
IR = M[ProgramCounter]
ProgramCounter = PC+4

Segnali di controllo:
Lettura in memoria = memread
Per scrittura nell'ir: IRWrite
Per indicare l'indirizzo da dove si deve leggere in memoria: IorD
Per incrementare il Program Counter: ALUSrca ALUSrcB, ALUop
Per salvare il nuovo valore del pc in pc: PCWrite

Passo 2: Decode (Tutte le istruzioni)

A prende i bit da 25 a 21 dell'ir
B prende quelli da 20 a 16

ALUOut prende pc+(sign-extended(ir[15:0]) shiftato tutto di 2

Segnali di controllo: ALUSrca, ALUSrcB, ALUOp utilizzati per un ipotetico calcolo di un indirizzo per una branch

---

Passo 3: Execute (per istruzioni lw e sw)

Ad ALUOut si assegna A + il sign extended di ir(15:0)

I segnali di controllo sono quindi ALUSrcA, ALUSrcB e ALUOp (che servono per il calcolo dell'indirizzo di memoria per le lw o sw

---

Passo 3: Execute (per istruzioni RType aritmetico logiche)

ALUOut prende AopB

I segnali sono sempre ALUSrcA, ALUSrcB e ALUOp per il calcolo aritmetico o logico

Passo 3: Execute (per le Beq)

Se A = B allora PC = ALUOut

Segnali di controllo: ALUSrcA, ALUSrcB e ALUOp per la comparazione di a e b
per quanto riguarda la scrittura del pc si usa pcwritecond e pcsource

Passo 3: Execute (per le Jump)

PC prende pc[31:28], ir[25:0] shiftato tutto di 2
I segnali di controllo son soltanto PcWrite e PCSource che servono per scrivere nel PC

Passo 4: Execute per le lw ed sw

MDR prende m[ALUOUT] OPPURE K[ALUOUT] prende B

Come segnali di controllo si hanno per indicare l'indirizzo di memoria: IorD
Per leggere dalla memoria in lw si usa memread e vbbbb memwrite per scrivere (ovvio)

Passo 4: per le istruzioni aritmetico logiche

Reg[15:11] = ALUOut
I segnali di controllo utilizzati sono:
RegWrite per scrivere nel register file
RegDist per indicare il registro su cui scrivere
Per scrivere ALUOut: MemToReg

Passo 5: Solo per la LoadWord

L'IR[20:16] = MDR
I segnali di controllo sono:
RegWrite per scrivere nel reg file
RegDest per indicare su che registro scrivere
MemToReg per scrivere il valore in memoria

Segnali di controllo a 1 bit

Actions of the 1-bit control signals

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register file destination number for the Write register comes from the rt field.	The register file destination number for the Write register comes from the rd field.
RegWrite	None.	The general-purpose register selected by the Write register number is written with the value of the Write data input.
ALUSrcA	The first ALU operand is the PC.	The first ALU operand comes from the A register.
MemRead	None.	Content of memory at the location specified by the Address input is put on Memory data output.
MemWrite	None.	Memory contents at the location specified by the Address input is replaced by value on Write data input.
MemtoReg	The value fed to the register file Write data input comes from ALUOut.	The value fed to the register file Write data input comes from the MDR.
IorD	The PC is used to supply the address to the memory unit.	ALUOut is used to supply the address to the memory unit.
IRWrite	None.	The output of the memory is written into the IR.
PCWrite	None.	The PC is written; the source is controlled by PCSource.
PCWriteCond	None.	The PC is written if the Zero output from the ALU is also active.

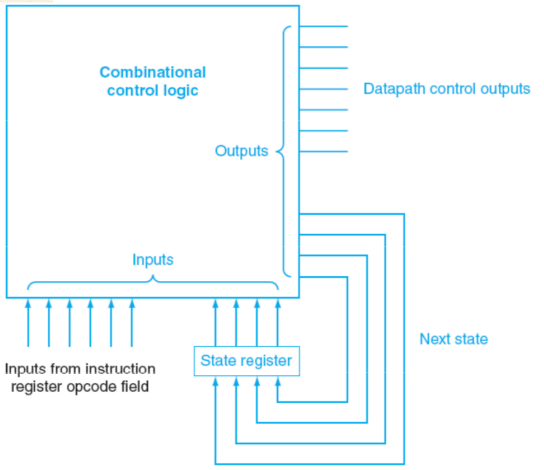
Segnali di controllo a 2 bit

Actions of the 2-bit control signals

Signal name	Value (binary)	Effect
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the B register.
	01	The second input to the ALU is the constant 4.
	10	The second input to the ALU is the sign-extended, lower 16 bits of the IR.
	11	The second input to the ALU is the sign-extended, lower 16 bits of the IR shifted left 2 bits.
PCSource	00	Output of the ALU (PC + 4) is sent to the PC for writing.
	01	The contents of ALUOut (the branch target address) are sent to the PC for writing.
	10	The jump target address (IR[25:0] shifted left 2 bits and concatenated with PC + 4[31:28]) is sent to the PC for writing.

FIGURE 5.29 The action caused by the setting of each control signal in Figure 5.28 on page 323. The top table describes the 1-bit control signals, while the bottom table describes the 2-bit signals. Only those control lines that affect multiplexors have an action when they are deasserted. This information is similar to that in Figure 5.16 on page 306 for the single-cycle datapath, but adds several new control lines (IRWrite, PCWrite, PCWriteCond, ALUSrcB, and PCSource) and removes control lines that are no longer used or have been replaced (PCSrc, Branch, and Jump).

Circuito sequenziale per il controllo



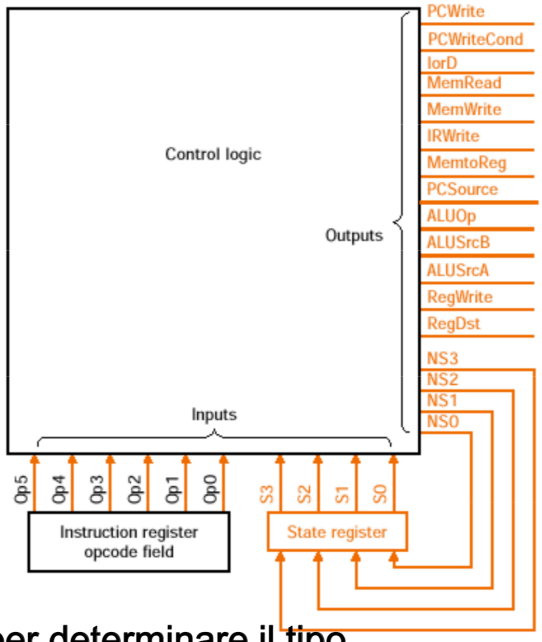
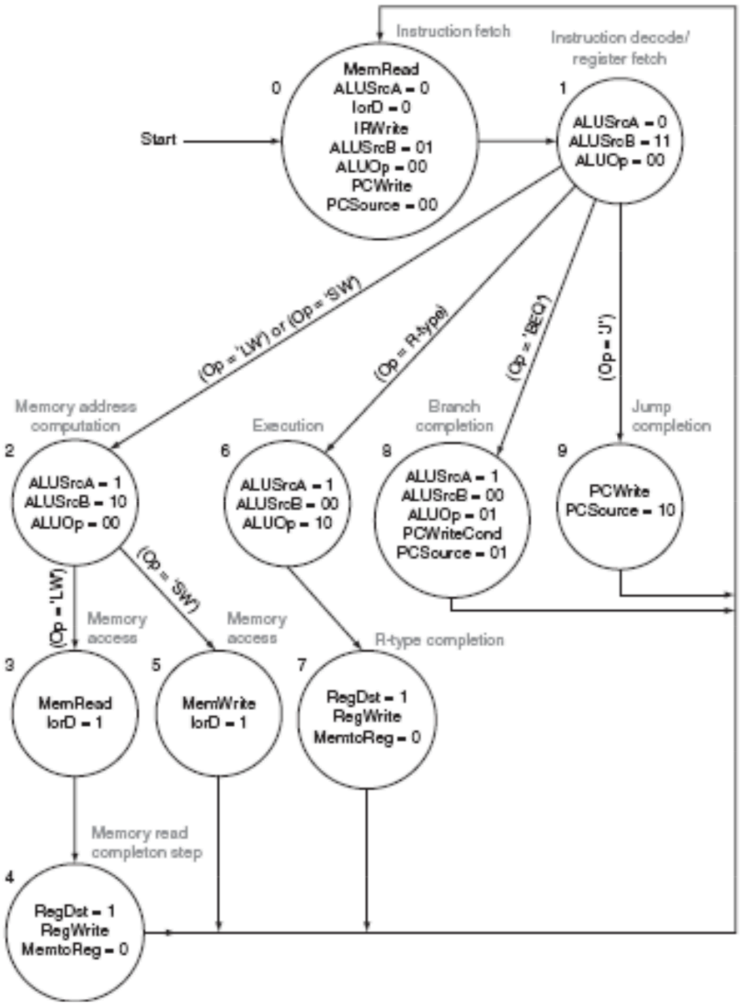


FIGURE 5.37 Finite state machine controllers are typically implemented using a block of combinational logic and a register to hold the current state. The outputs of the combinational logic are the next-state number and the control signals to be asserted for the current state. The inputs to the combinational logic are the current state and any inputs used to determine the next state. In this case, the inputs are the instruction register opcode bits. Notice that in the finite state machine used in this chapter, the outputs depend only on the current state, not on the inputs. The Elaboration above explains this in more detail.

- Controllo a due livelli: si usa OPCODE per determinare il tipo dell'istruzione e per R_type si usa anche il FUNC CODE
- State register memorizza lo stato corrente
- Blocco combinatorio (PLA) per il calcolo di NEXT_STATE e OUTPUT (memorizzate in ROM)

FSM



Gestione Eccezioni

L'eccezione è un imprevisto che avviene durante l'esecuzione di una istruzione, praticamente è sincrona con essa. Esempio: Scrivo una word di 4 bytes in uno spazio di 3 bytes

3 < 4 -> Overflow Exception. Nel mips esiste un modo preciso per individuare queste anomalie. OCCHIO ALLA DIFFERENZA TRA INTERRUPT ED ECCEZIONI

l'interrupt è ASINCRONO, è un evento esterno, qualcosa nato dall'utente stesso (Esempio la syscall, fornita in qtspim dal programma stesso)

In realtà è la macchina simulata che ha queste syscall automatiche. Infine abbiamo le trap che sono simili alle interrupt ma a prescindere da cosa sia un'eccezione.. Come si gestisce?

Gestione dell'eccezione

L'hardware RILEVA l'eccezione, viene eseguito un SOFTWARE che si avvia solo in caso è rilevata una determinata eccezione.

Come? In una determinata posizione nota all'hw c'è caricato questo software che viene eseguito di pacco dopo un'eccezione. Viene proprio sospeso tutto per risolvere l'eccezione. Questo software è caricato quindi di default dal loader di sistema in fase di accenzione

Hackerandolo puoi disattivarlo in modo che non venga eseguita la gestione dell'exception MA è importante specificare bene che il rilevamento dell'eccezione è hardware, la gestione è software Resta da chiedersi come fa a capire l'eccezione. C'è un registro o semiregistro del processore che viene scritto nel momento in cui si verifica un'eccezione

Questo registro è il registro cause il gestore e viene scritto all'eccezione, lo riceve il gestore che avrà mano sul PC. ed in base all'exception program counter capiamo in quale preciso momento è avvenuta l'eccezione. Il gestore può pure essere scemo e troncare tutta l'esecuzione da subito

Il gestore potrebbe essere anche stupido e troncare a prescindere l'esecuzione del programma

---

L'interrupt

L'interrupt è un tipo di eccezione, cosa si verifica però? Prima ancora del fetch c'è un unico gestore, che nel caso del MIPS in particolare è un unico codice possibile, e tra l'altro c'è da tenere in mente che era comunque mono thread, il gestore esegue il codice di soluzione e torna da dove il codice è stato lasciato.

Finita la gestione i registri vengono ripristinati i registri (ovviamende salvati prima) come per le eccezioni. Qua hai una grande banda passante ma latenza alta, e si deve fare in modo di evitare delle interrupt durante l'esecuzione della gestione dell'interrupt.

C'è un FLAH e una porta and con l'interrupt request, e in base all'AND eseguo o no.

Più si alza la banda passante più aumenta la latenza.

Gestione dell'input/output

Tornando al modello Cpu bus di sistema ram e periferiche, spostiamoci su queste ultime, c'è da dire che ogni periferica ha la sua struttura:

1. Interfaccia
2. Registro di stato
3. Registro per i dati

E' chiaro che i dati calcolati dal calcolatore finiscono nel registro dati e ATTRAVERSO le periferiche i dati sono comunicati all'esterno OPPURE ad una variabile di programma. In qualche maniera da lì i dati escon fuori insomma.

Periferica di uscita: Variabile di programma $\longmapsto$ esterno
Periferica di ingresso: esterno $\longmapsto$ Variabile di programma

Come viene interfacciato questo registro al calcolatore? Esempio: Se l'utente schiaccia un tasto, come finisce in memoria il codice ascii della lettera premuta?

Mappatura delle periferiche in memoria

O comunque periferiche mappate in uno spazio di indirizzamento generico distinto per l'ingresso ed uscita.

Cosa significa mappata?
Significa che si ha una struttura fisica della memoria fatta in un modo. Perchè si deve fare in modo che se si ha una determinata configurazione di bit che passa sul bus di controllo, allora si attiva solo quella che è mappata

Si fa in modo che il sistema di decodifica si attivi opportunamente quando ci sono più di uno di questi circuiti, deve attivarsi solo quello di riferimento.

Chiaro che la memoria sia sempre mappata in memoria, ma potrebbe comunque accadere che ci sia dello spazio da qualche altra parte, in un comunque generico spazio di indirizzamento

Per potere implementare questo tipo di selezione si utilizza una circuiteria di selezione, generalmente si parla di multiplexer o demultiplexer

Tecniche di gestione dell'I/O

Il controllo di programma:
C'è un registro di stato contenente una serie di bit, di cui un oè il bit Ready, che serve per capire se una periferica è pronta, in generale se uno construisce la periferica, fa in modo che questo accada così.

Dal punto di vista logico se(pronto) allora fai altrimenti no

E' una semplice branch if equals portandosi con una lw in $t0, e per esempio il valore del registro di stato. Se non è ready cicla all'infinito finchè non sarà pronto

Questo ciclo di attesa si chiama busy waiting cioè la cpu è occupata in questa attesa, si chiama Pooling questa cosa (ripeti finchè)

Se io son più veloce della mia periferica (cioè io utente che scrivo sulla tastiera per intenderci) allora vien contato solo l'ultima delle configurazioni, tipo non so faccio una parola, tiene le ultime lettere.

Quando i dati escono, di solito con loadword o qualcosa di simile, nel registro di stato si scrive 0 e se ne occupa la periferica.

Banda passante e latenza

La banda passante è la quantità di dati che vien passata per unità di dati, tipo gb al secondo, la frequenza insomma

Mentre la latenza è l'intervallo di tempo che passa da quando la periferica è pronta a quando prelevo i dati rendendo 0 il bit ready

Registro di stato della CPU

Ogni cpu ha il suo registro di stato che è ben diverso da quello delle periferiche, ed ha funzioni varie

Un sottoinsieme di bit appartenenti a questo registro praticamente serve per il mascheramento delle linee di interruzioni, poi ci sono macchine che hanno un registro di mascheramento.

Nel mips tutto questo accade nel registro di stato, ma questi bit chi ce li mette dentro?
Generalmente son tutti uno ma uno può interrompere un gestore interruzione, per interromperlo

Vettorizzazione

E’ l’alternativa della formula a singolo vettore, praticamente vengono testati i valori presenti nel registro cause, e con questa struttura si risparmiano le istruzioni del gestore che servono a capire quale periferica e quale causa, togliendo tutti gli if

Base + causa*4 (se son 4 bit) e ottieni l’indirizzo di inizio gestore causa che è un valore intero in pratica

La memoria cache

La ram è costituita da un largo quantitativo di locazioni ad accesso lento, ed una parte più piccola ad accessi più veloci.

C’è bisogno di fare in modo che in pratica questa appaia come una sola memoria costituita da queste due. E per fare questo viene in aiuto una memoria più veloce che si chiama CACHE.

Avendo una memoria grande e lenta, e volendo avere una memoria veloce con l’uso di una memoria più piccola, come gestiamo le cose nella memoria veloce?