Capitolo 5: Processore

Esercizio 9

La **MIC-1**, creata da Andrew S. **Tanenbaum**, è macchina virtuale che implementa la **IJVM** ovvero la Java Virtual Machine che opera esclusivamente su numeri interi. Il MIC-1 rappresenta un classico esempio di un sistema digitale basata su controllo microprogrammato anziché cablato.

L’architettura del MIC-1 è composta da vari componenti principali, in primis il **datapath**. Il percorso dati è, in generale, il sistema di circuiti che esegue le operazioni richieste dalle istruzioni del processore; nel caso del MIC-1 esso è formato dagli seguenti elementi:

* **Registri**
  + **MAR** (Memory Address Register): Questo registro contiene la posizione in memoria a cui accedere per svolgere una lettura o una scrittura di dati. Quando viene inserito un indirizzo nel MAR, i dati corrispondenti a quell’indirizzo vengono caricati nella MDR.
  + **MDR** (Memory Data Register): Questo registro contiene i dati che si leggendo o scrivono in memoria nella posizione data dal MAR.
  + **PC** (Program Counter): Questo registro contiene l'indirizzo della prossima istruzione da eseguire. Dopo l'esecuzione di un'istruzione, il PC viene incrementato di uno per puntare alla istruzione successiva.
  + **MBR** (Memory Buffer Register): Questo registro contiene la porzione da esaminare dell'istruzione corrente; questo registro è l’unico ad essere da 8 bit mentre tutti gli altri sono da 32 bit.
  + **SP** (Stack Pointer): Questo registro contiene un puntatore all’elemento in cima allo stack.
  + **LV** (Local Variable): Questo registro contiene un riferimento alla base sullo stack della **Local Variable Frame** del metodo attualmente in esecuzione. La Local Variable Frame contiene le variabili locali del metodo ed è sovrastata da l’**Operand Stack** il cui ultimo operando è proprio quello puntato da SP.
  + **CPP** (Constant Pool Pointer): Questo registro contiene l'indirizzo della base dall’area di memoria nota come **Constant Pool** la quale contiene le costanti, le stringhe e i puntatori del programma attivo.
  + **TOS** (Top Of Stack): Questo registro contiene una copia l'elemento in cima allo stack. Quando un nuovo valore diventa quello superiore sullo stack, il TOS viene aggiornato.
  + **OPC**: Questo registro viene utilizzato per contenere risultati intermedi delle operazioni.
  + **H** (Hold): Quando si esegue un'operazione aritmetica su due operandi, uno dei due viene prima trasferito nel registro H; l’altro invece viene preso direttamente dal bus al momento del calcolo.
* **Bus:**
  + **B**: Questo bus è utilizzato per trasferire i dati tra i registri e l'ALU. Tutti i registri menzionati in precedenza, ad eccezione di MAR, sono dotati di un segnale di controllo che gestisce la scrittura sul bus B. Si noti che in un dato istante, solo un registro può scrivere sul bus B.
  + **C**: Questo bus è impiegato per trasferire i dati tra lo shift register che contiene il risultato dell’ALU e i registri. Tutti i registri, tranne MBR, sono dotati di un segnale di controllo che permette la lettura dal bus C.
* **ALU** (Arithmetic Logic Unit): Questo modulo esegue le operazioni aritmetiche e logiche. Nella MIC-1, l'ALU ha sei linee di controllo, le quali determinato quale funzione eseguire e quali input utilizzare.
  + **F0** e **F1** selezionano la funzione dell'ALU: AND, OR, NOT o ADD.
  + **INVA** determina se eseguire o meno il complemento a uno, ossia la negazione, dell'operando proveniente dal registro H.
  + **ENA** attiva l'acquisizione del primo operando da H, altrimenti esso è zero.
  + **ENB** attiva l'acquisizione del secondo operando dal bus B, altrimenti esso è zero.
  + **INC** specifica se incrementare di uno il risultato ottenuto o mantenerlo invariato.

L'ALU ha anche due bit di output di stato: **N** e **Z** che indicando rispettivamente se il risultato dell'operazione è negativo oppure zero. I valori di questi flag sono memorizzati all'interno di due flip-flop, rendendoli facilmente accessibili.

Il risultato viene immagazzinato in uno shift register prima di essere trasferito sul bus C. Lo **shifter** è dotato di due segnali di controllo

* + **SLL8** (Shift Left Logical): Quando questa linea è attiva, lo shifter esegue un'operazione di shift a sinistra logico, spostando tutti i bit verso sinistra di 8 posizioni e riempiendo i bit più significativi con degli zeri.
  + **SRA1** (Shift Right Arithmetic): Quando questa linea è alta, lo shifter esegue un'operazione di shift a destra aritmetico ovvero sposta tutti i bit verso destra di 1 posizione e mantenendo il bit più significativo uguale al suo valore originale, zero o uno.

La MIC-1 dispone di due modalità di comunicazione con la memoria:

1. Un terminale a 32 bit controllato dai registri MAR e MDR: il MAR indirizza le word di dati, mentre il MDR gestisce la memorizzazione o l'acquisizione di tali dati.
2. Un terminale ad 8 bit controllato dai registri PC e MBR: il PC indirizza singoli byte di programma, e l'MBR è coinvolto nella lettura e scrittura di tali byte. Inoltre, oltre al segnale che gestisce la scrittura sul bus B, l'MBR riceve in ingresso un ulteriore segnale di controllo che determina se il byte deve essere trasmesso su B come un vettore di bit con segno o senza.

L'operazione di lettura o scrittura della memoria, effettuata con uno dei metodi sopra menzionati, si svolge in due fasi distinte. Inizialmente, si inserisce un indirizzo nei registri PC o MAR, avviando così la lettura o la scrittura. Tuttavia, tale operazione viene completata solo durante il ciclo di clock successivo, quando avviene l’effettivo caricamento dei dati da o verso il MBR o MDR.

Nel complesso, considerando che alcuni di essi sono mutualmente esclusivi, il datapath è guidato nel suo flusso di esecuzione da 24 segnali di controllo che vengono generati dal processore.

Nell'architettura della IJVM di Tanenbaum ad ogni istruzione ISA è associata una sequenza di microistruzioni da eseguire detta micro-routine. Tutte le possibili microistruzioni, chiamate anche **control word**, hanno una lunghezza fissa di 36 bit e sono memorizzate all’interno di una memoria ovvero la **Control Store** che può contenere fino a 512 di queste parole.

L'accesso alla Control Store avviene attraverso due registri chiave: il **MPC** (Micro Program Counter) e il **MIR** (Micro Instruction Register). Il MPC è un registro virtuale da 9 bit che stabilisce l'indirizzo da cui leggere la prossima microistruzione, mentre il MIR contiene la control word letta dalla posizione indicata in precedenza dal MPC.

Per semplicità, la posizione della prima microistruzione di un'istruzione è data proprio dal valore in binario dell'**opcode** corrispondente a quell'istruzione. Successivamente, nonostante le control word siano memorizzate “in ordine” nella Control Store, la loro esecuzione può seguire un percorso diverso da quello prestabilito in base a diversi fattori.

I primi 8 bit della microistruzione (**Addr**) nel registro MIR rappresentano l’ipotetico indirizzo della successiva microistruzione, mentre i successivi 3 bit (**JMPC**, **JAMN**, **JAMZ**) contengono codici di condizione per i salti. È in base a questi bit e ad altre informazioni provenienti dal datapath che si decide come aggiornare il MPC. Precisamente, vengono impiegati due blocchi logici:

1. **High bit**: Questo circuito stabilisce il bit più significativo di MPC in base al bit più significativo di Addr, ai bit JAMN, JAMZ e ai flag N e Z precedentemente presentati, secondo la seguente funzione.
2. **Jump logic**: Questo blocco calcola gli 8 bit meno significativi del nuovo contenuto di MPC utilizzando gli 8 bit meno significativi di Addr, l'intero contenuto del registro MBR e il codice JMPC. L'espressione da calcolare è riportata successivamente.

Se JMPC è alto, ci si aspetta che tutti i bit di Addr[1:8] siano zero; in questo caso quindi i bit MPC[1:8] sono posti proprio a MBR[0:7], che tipicamente rappresenta un codice operativo.

Invece, i restanti bit della control word, ovvero quelli successivi a JAMZ, rappresentano i segnali di controllo del percorso dei dati. Si osservi che i segnali che gestiscono la scrittura dei registri sul bus B sono codificati su 4 bit, poiché non sono necessari 9 segnali distinti, essendo questi mutualmente esclusivi.

Nell'immagine seguente è illustrata l'architettura complessiva del MIC-1.

Immagine che contiene testo, diagramma, Piano, mappa

Descrizione generata automaticamente

Per definire la sequenza di microistruzioni che implementa un'istruzione ISA, è possibile utilizzare il linguaggio **MAL** (Micro Assembly Language). Si noti che nel microcodice scritto in questo linguaggio simbolico il termine "fetch" indica la comunicazione con la memoria tramite i registri PC e MBR, mentre il termine "read" si riferisce alla comunicazione che coinvolge MAR e MDR.

L'istruzione **isub** è un'istruzione ISA che estrai due numeri interi del top dello stack, esegue la sottrazione tra di essi e fa il caricamento del risultato sulla cima dello stack. Chiaramente prima di eseguire isub, è necessario aver caricato almeno due operandi sullo stack, ad esempio, tramite due istruzioni di bipush. Il **bipush** esegue il push di un byte fornito come operando all’istruzione sullo stack.

Il microcodice MAL associato all'istruzione **isub** è il seguente.

isub = 0x5C:

MAR = SP = SP - 1; rd

H = TOS

MDR = TOS = MDR - H; wr; goto main

“0x5C” è il codice operativo dell’istruzione ISA.

1. **MAR = SP = SP - 1; rd**: Viene decrementato di 1 il registro e il valore risultate viene scritto anche in MAR. SP non punta più quindi alla cima dello stack ma all’elemento precedente che al momento contiene il primo operando della sottrazione e che alla fine conterrà il risultato. “**rd**” indica che è stata fatta partire un'operazione di lettura della memoria in corrispondenza della posizione data dal MAR, ovvero si sta leggendo il primo operando.
2. **H = TOS**: Viene copiato nel registro H il valore attualmente presente al top dello stack, corrispondente al secondo operando, poiché il registro TOS non è stato ancora aggiornato.
3. **MDR = TOS = MDR - H; wr; goto main**: Effettua la sottrazione tra i registri MDR (il primo operando poiché a questo punto si è conclusa la lettura) e H (il secondo operando) e il risultato viene salvato sia in MDR che in TOS. “**wr**” indica che è in corso un'operazione di scrittura del valore di MDR, ovvero il risultato della differenza, nell’indirizzo ancora mantenuto in MAR che chiaramente non è stato necessario modificare. Infine, l'istruzione "**goto**" consente di eseguire un salto incondizionato all'etichetta "main".

Per simulare l’istruzione isub è stato modificato il file “program.ajvm” come segue.

.main

.var

a

.endvar

BIPUSH 0xA

BIPUSH 0x5

ISUB

ISTORE a

HALT

.endmethod

In questo programma, vengono caricati i valori 0xA e 0x5 sulla pila tramite le istruzioni BIPUSH. Successivamente, l'istruzione ISUB sottrae il secondo valore dal primo, modificando così lo stack da [0xA, 0x5] a [0x5]. L'istruzione **ISTORE** memorizza quindi il risultato, 0x5, nella variabile “a”. Infine, l'istruzione **HALT** termina l'esecuzione del programma.

Tra i testbench forniti, l'unico che richiede modifiche è "processor\_tb.vhd"; è infatti necessario adattare il processo “wavegen\_proc”.

-- Waveform generation

wavegen\_proc: process

begin

wait until clk = '1';

wait for 2 ns;

reset <= '1';

wait for 10 ns;

reset <= '0';

wait until mem\_instr\_addr = x"0000000A" and mem\_data\_we = '1';

assert mem\_data\_out = x"00000005" report "Bad calculated value" severity failure;

wait until mem\_instr\_addr = x"0000000B";

end\_run := true;

wait;

end process wavegen\_proc;

Le immagini sottostanti mostrano i risultati positivi dei test

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

L'istruzione **swap** effettua la rimozione di due numeri dalla cima dello stack e successivamente li reinserisce sulla pila, ma in ordine inverso. Il microcodice MAL associato all'istruzione **swap** è scritto successivamente.

swap = 0x5F:

MAR = SP - 1; rd

MAR = SP

H = MDR; wr

MDR = TOS

MAR = SP - 1; wr

TOS = H; goto main

“0x5F” è l’opcode dell’istruzione ISA.

1. **MAR = SP - 1; rd**: Viene letto dalla memoria il valore nella posizione immediatamente sotto la cima dello stack. Solo successivamente il dato letto, ovvero il primo dei due elementi da scambiare, viene memorizzato nel registro MDR.
2. **MAR = SP**: Aggiorna il registro MAR in modo che punti al top dello stack, ovvero al secondo dei due numeri da scambiare.
3. **H = MDR; wr**: Viene scritto il valore contenuto in MDR, quindi il primo numero, nel registro temporaneo H. Si fa, poi, partire la lettura del secondo numero dato che ora, poiché il registro MAR è stato opportunamente configurato a puntare al TOS.
4. **MDR = TOS**: Carica l’elemento attualmente presente nel registro TOS, ovvero il secondo numero, in MDR.
5. **MAR = SP - 1; wr**: Aggiorna MAR a puntare nuovamente alla posizione sotto la cima dello stack. Si dà il via a un’operazione di scrittura dei dati in MDR, ossia il secondo valore, all’indirizzo dato da MAR quindi nella posizione sotto il top.
6. **TOS = H; goto main**: Si scrive il contenuto di H (il primo elemento) sul top dello stack completando così lo swap. Infine, c'è un salto all'etichetta main.

Per emulare l'istruzione swap, il file "program.ajvm" riscritto nel seguente modo.

.main

.var

a

.endvar

BIPUSH 0x1

BIPUSH 0x2

SWAP

ISTORE a

HALT

.endmethod

Prima dell’operazione di scambio la pila degli operandi è [0x1, 0x2], dopo diventa [0x2, 0x1]; quindi alla fine del programma la variabile “a” contiene 0x2.

Come in precedenza, si modifica il processo con label “wavegen\_proc” del file "processor\_tb.vhd" al fine di testare specificamente questo aspetto.

-- Waveform generation

wavegen\_proc: process

begin

wait until clk = '1';

wait for 2 ns;

reset <= '1';

wait for 10 ns;

reset <= '0';

wait until mem\_instr\_addr = x"0000000A" and mem\_data\_we = '1';

assert mem\_data\_out = x"00000001" report "Bad calculated value" severity failure;

wait until mem\_instr\_addr = x"0000000B";

end\_run := true;

wait;

end process wavegen\_proc;

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Software multimediale, software

Descrizione generata automaticamente

L’istruzione **iand** fa il pop di due elementi dello stack, esegue l’operazione logica di AND tra i due e fa il push del risultato sulla pila.

iand = 0x7E:

MAR = SP = SP - 1; rd

H = TOS

MDR = TOS = MDR AND H; wr; goto main

Adesso si modifica quest'istruzione, nel file “ajvm.mal”, in modo che anziché effettuare una AND, venga eseguita un'operazione di OR.

iand = 0x7E:

MAR = SP = SP - 1; rd

H = TOS

MDR = TOS = MDR OR H; wr; goto main

1. **MAR = SP = SP - 1; rd**: L’SP viene modificato in modo tale da puntare all’elemento prima del top dello stack; l’indirizzo ricavato viene scritto anche nel registro MAR e si fa partire una lettura in questa posizione che contiene uno dei due operandi.
2. **H = TOS**: Il valore che al momento si trova sulla cima della pila, ovvero l’altro operando, viene memorizzato in H.
3. **MDR = TOS = MDR OR H; wr; goto main**: Al posto dell'operazione logica di AND, viene eseguita un'operazione di OR tra gli elementi presenti nei registri MDR (che ora contiene il primo operando) e H. Il risultato viene scritto sia in TOS che in MBR e si dà il via alla scrittura di questo dato nell’indirizzo ancora memorizzato all’interno di MAR. Alla fine si fa un salto incondizionato al main.

Il programma che permette di testare questa modifica è riportato qui sotto.

.main

.var

a

.endvar

BIPUSH 0x1

BIPUSH 0x0

IAND

ISTORE a

HALT

.endmethod

È necessario verificare che la variabile "a" contenga 1 anziché 0. Il process “wavegen\_proc” deve quindi essere scritto nel seguente modo.

-- Waveform generation

wavegen\_proc: process

begin

wait until clk = '1';

wait for 2 ns;

reset <= '1';

wait for 10 ns;

reset <= '0';

wait until mem\_instr\_addr = x"0000000A" and mem\_data\_we = '1';

assert mem\_data\_out = x"00000001" report "Bad calculated value" severity failure;

wait until mem\_instr\_addr = x"0000000B";

end\_run := true;

wait;

end process wavegen\_proc;

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente