Memòria de l’ampliació del processador

Projecte d'Enginyeria de Computadors – Primavera 2023

Índex

Sistema Operatiu

Introducció

L'objectiu d'aquesta part de l'ampliació era programar un sistema operatiu que tingués la capacitat d'alternar l'execució entre dos programes donat un *quantum*, calculat a partir de les interrupcions de *timer*. A més, el sistema operatiu ha de gestionar les diferents excepcions i interrupcions que siguin necessàries.

Característiques

El nostre sistema operatiu es troba al fitxer *OS.s* i conté: la funció *main*, encarregada d'inicialitzar l'estat del processador per al seu bon funcionament (és a dir, amb interrupcions activades) i de passar a mode usuari; la rutina de servei general (RSG), que gestiona totes les excepcions i interrupcions necessàries per al funcionament correcte dels programes a ser executats; i la rutina de canvi de context (RCC), que és l'encarregada d'alternar els dos processos que s'executen simultàniament: *corre-letras* i *fibonacci*. A aquesta rutina s'hi accedeix a partir d'una interrupció de rellotge.

Implementació i decisions de disseny

La primera decisió de disseny va ser com organitzar les regions de memòria d'usuari. Tenint dos processos i 0x8000 bytes, hem dividit aquest espai en blocs de 0x2000 bytes. El primer bloc, a partir de 0x0000, s'emmagatzema la regió de dades de *corre-letras*. A la direcció 0x2000, comença el codi de *corre-letras* i la pila d'usuari d'aquest procés, que creix cap a direccions més baixes. La mateixa distribució s'aplica per a *fibonacci*: a 0x4000 comença la regió de dades del procés, i a 0x6000 el codi del programa. De nou, la pila creix cap a direccions més baixes a partir de 0x6000. A partir de 0x8000 comença la regió de dades del sistema operatiu, i a 0xC000, el codi del sistema.

La gestió de les piles de sistema és la següent: per a simplificar el màxim possible la seva gestó, i donat que en el nostre escenari l'únic que fem és salvar, i posteriorment restaurar, l'estat, hem decidit que la pila de sistema comença allà on es troba la pila d'usuari un cop s'entra a mode sistema. La implementació d'aquest sistema l'hem fet amb dues variables globals, una per a cada programa, que conten les direccions a la pila just després de salvar el context del seu respectiu procés, de manera que l'únic que hem de fer per a canviar de context és accedir a la variable que conté la posició de memòria corresponent a la pila de l'altre procés, i substituir el valor actual de R7 per el valor llegit, on podrem desempilar el context i sortir de la rutina de canvi de context tal com hem entrat en el canvi anterior. Evidentment, abans de canviar el valor de R7 caldrà guardar el valor actual a la posició de memòria corresponen del procés actual, per a que el procediment es pugui repetir posteriorment en sentit contrari.

A l'hora de realitzar un canvi de context, primer es passa per la RSG, alhora que s'entra a mode sistema. El primer que es fa a la RSG és emmagatzemar el context, això és: empilar els registres des de R0 fins a R6, ja que R7 és el punter a la pila, la modificació del qual forma part del canvi de context; i els registres S0 (la paraula d'estat), S1 (la direcció de retorn al programa) i S2 (informació addicional sobre l'excepció que ha provocat l'entrada a mode sistema). Si la RSG determina que es tracta d'una interrupció de *timer*, passem a la seva subrutina, on primer incrementem en 1 la variable

global *tics_timer*, utilitzada pel programa *corre-letras*, i posteriorment saltem **sempre** a la rutina de canvi de context. Això equival a tenir un *quantum* d'un tic decisió que hem pres per senzillesa. Dins de la RCC, es determina en quin procés s'està en base al valor del registre S4, on un 0 indica *corre-letras*, i un 1, *fibonacci*. Aquest registre s'inicialitza a 0 donat que el nostre sistema operatiu comença executant *corre-letras*. Un cop determinat el procés actual, la rutina sap on ha d'emmagatzemar el valor de R7, és a dir, del punter a la pila del procés actual, i d'on ha de llegir el valor del punter a la pila del procés al que s'ha de canviar. Es realitzen aquesta escriptura i aquesta lectura, i posteriorment es canvia el valor de S4 al que correspongui. Fet això, es torna al final de la RSG, on es desempila l'estat del nou procés (que figurava a la seva pila per l'anterior canvi de context), i es retorna a mode usuari per a continuar amb l'execució del programa.

La inicialització del sistema operatiu és molt senzilla: el que ens cal és inicialitzar la pila del segon programa per tal de preparar el primer canvi de context. Per fer-ho, guardem el que després seran els registres R0 a R6 (que guardarem inicialitzats a 0), i els registres que correspondran a S0, S1, i S3. En aquest cas, S0 contindrà el valor 0x2 (la PSW amb interrupcions activades), S1 l'adreça d'inici del programa 2 (0x6000), i el valor d'S3 ens és indiferent. Després fem que R7 (recordem que s'utilitza com a *stack pointer*) apunti a la pila del primer programa, definim S1 com l'adreça on comença el primer programa (0x2000) i finalment saltem a mode usuari mitjançant una instrucció de *reti*, que ens farà començar l'execució d'aquest.

El tractament de la resta d'excepcions i interrupcions no difereix gaire de la primera meitat del tractament d'un canvi de context: un cop dins de la RSG, es determina de quina excepció o interrupció es tracta, i es salta a la corresponent subrutina. Cal dir que, tot i tenir implementades les crides a sistema i totes les interrupcions, cap fa res a part de continuar l'execució, excepte la interrupció de *timer* i la de teclat, que són les úniques que es fan servir en els dos programes a executar, ja que la resta de control amb els perifèrcs (concretament en el cas de *fibonacci*) es fa per *polling*. La gestió de les excepcions tampoc ha estat implementada amb detall, es deixa a futures implementacions del sistema operatiu a ser definida pel programador.

Ampliació Hardware

Introducció

L'objectiu d'aquesta part de l'ampliació era implementar una unitat de càlcul específica per a tractar amb números de coma flotant. Aquesta unitat permet efectuar operacions aritmètiques i lògiques en nombres en format de coma flotant.

Característiques

Aquesta unitat ha estat implementada segons l'estàndard de l'assignatura, definit al document SISA-F. Dels 16 bits de les nostres dades, tenim un de signe, sis d'exponent i nou de mantissa. El bit de signe es codifica 0 per als valors positius i 1 per als negatius, l'exponent es codifica en binari excés 31 i la mantissa en binari natural, normalitzada al primer bit significatiu diferent de 0, amb coma a la dreta i bit ocult (1+0,f sent f la mantissa de 9 bits emmagatzemada).

Implementació i decisions de disseny

El primer que es va decidir va ser la codificació d'un nou banc de registres específic per coma flotant. Aquest té la mateixa estructura que el banc de registres d'enters. A causa d'aquest nou banc, s'han ampliat els senyals de control per discriminar les sortides del banc de registres segons les operacions. Per repartit millor la feina es va dividir la FPU en tres mòduls, sumes/restes, multiplicacions/divisions, comparacions.

Pel que fa al codi de mul/div, està extret del fòrum EDABoard, els quals han estat adaptats al nostre model de coma flotant.
(https://www.edaboard.com/threads/floating-point-divider-vhdl-code-or-full-architecture.184079/#post-1023975)(https://www.edaboard.com/threads/floating-point-multiplication-using-vhdl.52628/post-1023992)

El codi del sumador ha estat extret de Github (https://github.com/prashal/fp_adder) i també ha estat adaptat als nostres estàndards. Per a restar es fa servir el mateix mòdul, però canviant el signe de Y abans d'executar l'operació. Per a un correcte funcionament de la normalització de l'exponent en sumes i restes s'ha hagut d'augmentar el nombre d'estats en el processador, on s'ha afegit l'estat FLOAT. Aquest només es fa servir quan es necessiten cicles extres de càlcul en operacions de suma o resta de coma flotant. El següent estat ve establert per un senyal de control que indica si s'ha acabat l'operació de coma flotant.

En el mòdul de comparacions, a cada cicle es calculen els tres possibles flags; igual, menor a, menor igual i es multiplexen segons es necessiten.

S'han implementat les excepcions de divisió per zero i overflow en coma flotant, aquesta última sent enmascarable amb el bit 2 del registre 7 de sistema, tal com especifica SISA-F. Com a detall, dins del mòdul exc.vhd, s'ha invertit l'ordre de consulta dels flags d'excepcions Divisió entre zero en float i Overflow de float per un correcte funcionament de les excepcions.

SISA-F és contradictori. A la pàgina 21 s'especifica que totes les operacions de coma flotant estan sota el codi d'operació 1001, i tenen com a font i destí registres float. Una mica més endavant, a la

pàgina 24 i 26 se separen en dos codis d'operació, 1001 per sumar, restar, multiplicar i dividir, amb font i destí float i 1010 per comparacions amb font float i destí enter.

Hem decidit seguir el estàndard definit a la pàgina 21 i que totes les operacions tinguin codi 1001 i registres font i destí float. Si després es vol consultar el resultat de les comparacions s'haurà de fer un STF i LD.

Ús

L'ús d'aquestes noves instruccions està definit a la pàgina 21 de SISA-F al web de l'assignatura.

Jocs de prova

S'han proposat un seguit de jocs de proves dividits en els tres diferents moduls. Cadascun prova la funcionalitat per separat de diferents tipus d'instruccions.

El joc de proves comp_float.s comprova diferents casos, com ara NaN = NaN, -inf < inf, 0 < 2, etc. A cada comparació es consulta el resultat i s'incrementa el comptador si la comparació ha estat certa, que es mostra per el 7-segments. Hi han 20 comparacions certes, o 0x14.

div_float.s i mul_float.s comproven diferents multiplicacions i divisions amb nombres normals o especials com ara inf*5.2265625 i mostra el resutlat pel 7-segments. Cada exemple s'executa quan els interruptors estan en un cert estat. Començant per interruptos = 0 i per ordre, si es va activant nomes un cada cop, primer SW0, despres es desactiva i s'activa nomes SW1, etc. Pels leds vermells es mostra el codi d'excepcio si es el cas que n'hagi saltat alguna.

Els jocs de prova del sumador-restador estan dividits en dos grups: sumes i restes de coma flotant. Per a cadascun dels casos s'ha provat casos amb operacions regulars i després casos on s'activi la excepció per overflow de coma flotant.

Es pot comprovar amb SignalTap els valors del registre final és l'esperat en els casos amb operacions regulars i el valor del senyal de overflow en els casos extrems.