Pràctica 2

Processadors vectorials (I)

1 Introducció

L'estudiant es va familiaritzar en el curs d'Arquitectura *de Computadors* amb els processadors segmentats i en particular amb el processador DLX desenrotllat per John L. Hennessy i David. A. Patterson. Una extensió natural de la segmentació del DLX consistix a introduir l'arquitectura vectorial DLXV, mantenint l'arquitectura DLX com la part escalar de l'arquitectura DLXV i la compatibilitat amb el seu joc d'instruccions.

En la present pràctica i la següent s'analitzarà la bondat de l'arquitectura vectorial DLXV i els seus problemes per mitjà de l'execució de programes escrits en ensamblador DLXV en el simulador xdlxvsim. És pretén que l'estudiant conega els fonaments dels arquitectures vectorials i seguisca capaç d'optimitzar el codi ensamblador per a evitar cingles i augmentar el rendiment de la màquina vectorial.

2 L'arquitectura DLXV. Revisió

L'arquitectura de processador vectorial que es va a utilitzar s'anomena DLXV (ja que la seua part escalar és la del DLX) i la seua part vectorial és l'extensió del DLX. Se suposa l'estudiant familiaritzat amb l'arquitectura escalar DLX i els conceptes bàsics de segmentació.

Les principals característiques de l'arquitectura DLXV són:

1. **Registres vectorials**. El DLXV posseïx 8 registres vectorials i cada registre vectorial emmagatzema 64 dobles paraules (una doble paraula consta de 64 bits). Cada registre vectorial té dos ports de lectura i un d'escriptura. A més d'estos registres existixen 3 registres especials VLR, MVL i VMR. La primera marca la longitud del vector en l'operació vectorial (la longitud pot canviar). El segon és fix i indica el valor màxim de VLR (per a DLXV serà 64). El tercer és la màscara vectorial booleana de longitud MVL que s'utilitzarà per a fer operacions amb instruccions condicional i matrius disperses.
2. **Unitats funcionals vectorials**. Cada unitat està completament segmentada i pot iniciar una nova operació en cada cicle de rellotge (després d'una certa latència). Es necessita una unitat de control per a detectar els conflictes estructurals i les dependències de dades. El DLXV posseïx diversos tipus d'unitats funcionals que són: Unitat de càrrega/emmagatzematge vectorial, la màxima/resta en punt flotant, multiplicació en punt flotant, divisió en punt flotant, unitat d'operacions senceres i unitat d'operacions lògiques. El següent quadro definix les latències inicials de les unitats funcionals.

Operació	Latència
Càrrega vectorial	12
Suma vectorial	6
Multiplicació vectorial	7
Divisió vectorial	20

Les instruccions vectorials, quan entren en la unitat segmentada no es distingixen de les altres, de fet les etapes IF i ANEU de la màquina DLX són idèntiques. Només quan van a emetre's (passar a EX) es detecta que són instruccions vectorials i, en este cas, passen a una etapa d'identificació d'instruccions vectorials: IV

Perquè una instrucció vectorial puga passar de l'etapa ANEU a la IV han de comprovar-se les dependències amb les dades escalares o registres en coma flotant. Una vegada lliures de dependències estes dades són llegits i romanen associats a la instrucció vectorial, la qual passa a l'etapa IV.

En l'etapa IV poden estar un màxim de N instruccions vectorials encolades, si arriba una instrucció vectorial N+1 llavors esta haurà d'esperar en ANEU, detenint per tant tota la segmentació. El valor de N és la longitud de la cua d'instruccions vectorials i és un paràmetre de la màquina vectorial. En el cas de DLXV serà de 5.

Les instruccions vectorials que es troben en IV són emeses per orde, és a dir; l'etapa IV es comporta com una cua FIFO, de manera que una instrucció vectorial no pot avançar a una altra en esta etapa. Per tant, la instrucció vectorial que pot emetre's potencialment és aquella que va arribar abans a l'etapa IV. Perquè siga possible emetre una instrucció vectorial han de comprovar-se:

1. Dependències estructurals; amb la unitat d'operació vectorial necessària i amb els ports de lectura/escriptura.
2. Dependències de dades.

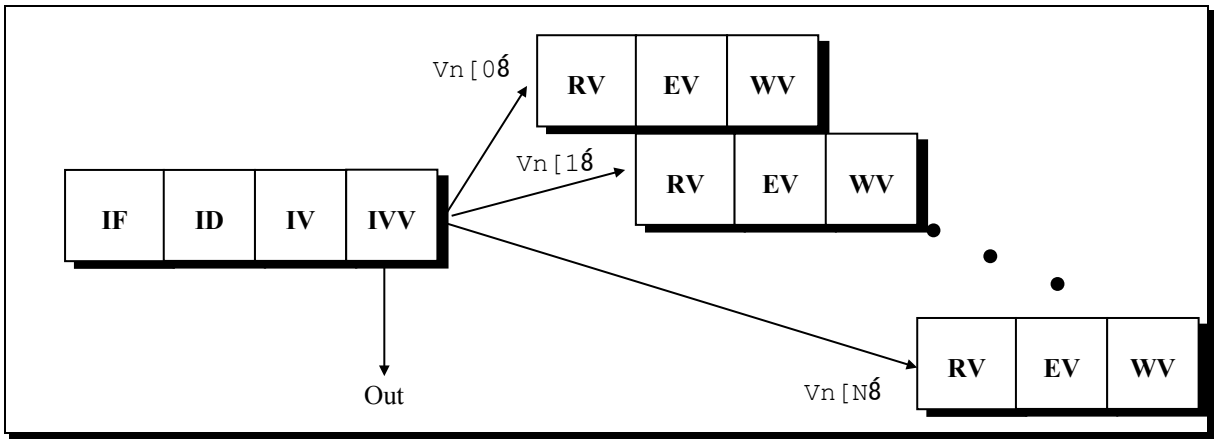
Si s'emet amb èxit una operació vectorial, s'intentarà emetre la següent de l'etapa IV i així successivament fins que no quede cap instrucció en IV o bé es produïsca alguna dependència i no puga emetre's la corresponent instrucció vectorial. Una vegada emesa una instrucció vectorial, esta passa a una etapa IVV. En realitat esta etapa conté les instruccions vectorials que es troben executant-se. En cada cicle de rellotge, per cada una de les instruccions que hi ha en IVV es llança una operació d'un element individual del vector. La dita operació consta de tres etapes RV, EV i WV. Per descomptat, en estes etapes no cal comprovar cap dependència, ja que estes ja han sigut comprovades al emetre la instrucció vectorial.

Este esquema és el mateix per a les operacions de càrrega/emmagatzematge vectorial, en les quals en l'etapa EV es du a terme l'accés a memòria. La direcció efectiva la calcula la pròpia unitat de càrrega/emmagatzematge vectorial, la qual haurà de distingir entre els tres tipus d'accés: seqüencial, amb *stride* i amb vector d'índex.

En l'etapa d'escriptura cal tindre en compte si l'element que es va a escriure resulta emmascarat pel `VMR` associat a la instrucció i en este cas inhibir la dita escriptura.

Quan s'aconseguix l'últim element del vector, definit pel valor `VLR` associat a la instrucció vectorial; llavors la dita instrucció acaba la seua execució i és eliminada de l'etapa IVV.

L'esquema de fases d'una instrucció vectorial és, per tant, el següent:



Podem observar que la instrucció vectorial acaba en l'etapa IVV, mentres que les instruccions d'operacions sobre cada element del vector acaben en WV. Es generen un nombre igual a `VLR` d'operacions elementals per cada instrucció vectorial.

3 Desenrotllament de la pràctica

Una vegada revisats els principals conceptes de l'arquitectura vectorial DLXV es va a experimentar amb diversos segments de codi en ensamblador del DLXV per a observar les principals característiques del processador.

Exercici 1

Observa l'exemple 1 que es mostra a continuació. En el es realitza una resta vectorial sota el control de la màscara vectorial VMR. Genera un fitxer amb el codi proposat i carrega-ho en el simulador. **Realitza una simulació pas a pas del programa des de la direcció `inici`** observant el llit, l'execució d'instruccions i la zona de memòria corresponent utilitzant les finestres adequades del simulador.

```
;Exemple vectorial 1
;Modificacio de la longitud del vector i execucio condicional
 d'instruccions de carrega emmagatzematge


        .data 0
        .global a
a:      .double 100, 0, 100, 0, 100, 0, 100, 0, 100, 0
        .double 100, 0, 100, 0, 100, 0, 100, 0, 100, 0
        .global b
b:      .double 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
        .double 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
        .global zero
zero:   .double 0

inici:   add     r10,r0,a   ;r10 conté la direcció del vector A
         add     r11,r0,b   ;r11 conté la direcció del vector B
         add     r1,r0,#20  ;20 -> r1
         movi2s  vlr,r1     ;20 -> vlr Modifiquem la longitud del vector
         lv      v1,r10     ;Càrrega vector A en V1
         ld      f0,cero    ;Càrrega un zero en coma flotant en F0
         snesv   f0,v1      ;Posa VM a 1 sí V1(i) diferent de F0
         lv      v2,r11     ;Càrrega en V2 els elements de B que toquen
         subv    v1,v1,v2   ;Resta sota el control de la mastegara
         sv      r10,v1     ;Emmagatzema el resultat en A

         trap    #0            ;Fi del programa vectorial
```

Comprova les detencions que es produïxen en el pipeline. Per què es produïxen estes detencions? Realitza una simulació del mateix codi anterior però esta vegada amb l'opció d'encadenament de les instruccions vectorials de la màquina. Quina diferència es produïx en el nombre de detencions? De que dependran les detencions en este segon cas?

Exercici 2

Es proposa a l'alumne que escriba una rutina d'ensamblador DLXV que realitze l'operació `Z=a*X+Y` amb el menor nombre de cicles possible sent X e Y vectors de 157 elements numèrics en coma flotant de doble precisió i `a` la constant ✓emmagatzemada com a número de doble precisió. La configuració del processador DLXV serà la indicada per defecte en el simulador. Els vectors X e Y estaran emmagatzemats en les direccions x e y. El vector Z ho emmagatzemarem a continuació d'estos i abans del codi d'instruccions. Reservarem l'espai per al vector Z amb la directiva de compilació:

```
.space 1256
```

Com els vectors tenen més elements que els registres vectorials de la màquina, caldrà repetir els càlculs amb ajuda d'un bucle o desenrotllant les instruccions. En este cas l'alumne **ha d'utilitzar un sol bucle com a única solució a este problema**, ni tan sols es pot posar part del vector fora del bucle.

Es recorda que en alguns processadors RISC cal prendre precaucions amb els bots. En el cas del dlxvsim cal posar instruccions **nop** després d'una instrucció de bot condicional (**bnez**) per a evitar que s'execute la següent instrucció de forma involuntària.

Exercici 3

Anem ara a continuar analitzant la dependència de les detencions amb la configuració de la màquina vectorial. Per a això executa el següent codi amb l'opció d'encadenament.

```
;Exemple Vectorial 2

        .data 0
        .global a
a:      .double 3.14159265358979
        .global x
x:      .double 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
        .double 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
        .global y
y:      .double 100, 100, 100, 100, 100, 100, 100, 100, 100, 100
        .double 100, 100, 100, 100, 100, 100, 100, 100, 100, 100

inici:   add     r10,r0,x      ;r10 conté la direcció del vector X
         add     r11,r0,y      ;r11 conté la direcció del vector Y
         add     r1,r0,#20     ;20 -> r1
         movi2s  vlr,r1        ;20 -> vlr
         ld      f0,a          ;Càrrega escalar a
         lv      v1,r11        ;Càrrega vector X
         lv      v2,r10        ;Càrrega vector Y
         multsv  v3,f0,v1      ;Multiplicació vector*escalar
         addv    v4,v1,v1      ;Suma vectors
         sv      r11,v1        ;Emmagatzema el resultat

         trap    #0            ;Fi del programa
```

A què es deuen les detencions que es produïxen en l'anterior codi? Com es podria reconfigurar el processador per a evitar estes detencions?

4 Lectures addicionals i agraïments

Els fonaments dels processadors vectorials (i en particular del DLXV) estan perfectament documentats en *Arquitectura de Computadors. Un enfocament qualitatiu*. de J. L. Hennessy i D. A. Patterson. D'este llibre s'ha extret pràcticament tota la informació per a realitzar les dos pràctiques de processadors vectorials d'esta assignatura. Es recomana una lectura detallada del capítol 7 sobre processadors vectorials i DLXV.

El simulador xdlxvsim ha sigut realitzat per Gràcia Sánchez Carpena com a projecte fi de carrera sota la direcció del Dr. Pedro López Rodríguez, professor titular de la Universitat Politècnica de València. El professor López ha desenrotllat també el simulador de DLXV per a entorn Windows i este està disponible en la seua pàgina de *web* en la Universitat Politècnica de València.

Apèndix A: El simulador xdlxvsim

El simulador està compilat per a un entorn de X-windows davall linux, d'esta manera es deuran d'arrancar les X en la partició de RedHat5.1 que ha sigut utilitzada en les practiques anteriors. Una vegada es tinga l'entorn de X s'haurà d'executar: `xdlxvsim`. A l'arrancar l'aplicació apareixerà una finestra amb el menú principal:

Àrea del menú	**Botons de control**	**Botons ràpids**
File	Pipeline	Step
Execute	Clock Cycle Diagram	Run
Registers	Virtual FP Stages	
Memory	Floating Point Stages	
Configuration	Vectorial Stages	
Statistics	Vectorial Lloeu Stages	
Breakpoints	Vectorial Registers	
About		

Menú File

Si seleccionem el menú File tenim les opcions següents:

- Reset DLXV ® *Es posen a zero tots els registres del processador i també les estadístiques*.
- Reset ALL ® *Igual que l'anterior i a més es neteja la memòria.*
- Lloeu ® *Es carrega un programa en la memòria del DLXV des d'un fitxer ASCII.*
- Quit ® *Ix del simulador xdlxvsim.*

Menú Execute

Dins del menú Execute estan les opcions següents:

- Single Cycle ® *Executa un cicle de rellotge a partir de la direcció que preguntarà després.*
- Múltiple Cycles ® *Executa un cert nombre de cicles a partir d'una direcció.*
- Run ® *Executa a partir de la direcció donada fins que trobe una excepció.*
- Run To ® *Executa a partir de la direcció donada fins a una altra direcció.*

Menú Registers

En el menú Registers es tenen les opcions següents:

- Internal ® Mostra *els valors dels registres interns del processador.*
- Special ® Són *els valors dels registres especials del processador (PC, VLR, VM...)*
- Integer ® Mostra *els valors dels registres sencers* `r0` - `r31 del DLX.`
- Simple FP ® Mostra *els valors dels registres en coma flotant de simple precisió* `f0` - `f31`
- Double FP ® *Mostra els valors dels registres en coma flotant de doble precisió* `f0-f30`
- Vectorial ® *Mostra una finestra amb 8 botons. Un per a cada registre vectorial.*

Menú Memory

Si seleccionem el menú Memory veiem les opcions següents:

- Display ® Amb *esta opció es pot veure el contingut de la part de memòria que es desitge.*
- Change ® *Es pot veure i canviar el contingut de la part de memòria que es desitge.*
- Symbols ® Amb *esta* opció *es poden veure els símbols associats a direccions de memòria*

Menú Configuration

Dins del menú Configuration es tenen les opcions següents:

- Memory ® *Configura la grandària de la memòria i el nombre de mòduls de memòria.*
- Floating Point ® *Configura les unitats d'operació de coma flotant.*
- Vectorial ® *Configura les unitats d'operació vectorials.*
- Machine ® *Configura les opcions de funcionament del processador (planificació, etc..)*

Menú Statistics

En el menú Statistics tenim les opcions següents:

- Stalls ® *Mostra estadístiques de parades (vectorials, senceres i de punt flotant).*
- Operation Count ® *Compte les operacions executades.*
- Conditional Branches ® *Mostra les estadístiques sobre els bots condicional.*

Menú Breakpoints

Dins del menú Breakpoints tenim les opcions següents:

- Display ® Mostra *els punts de ruptura establits.*
- Set ® Establix *un punt de ruptura en la direcció donada.*
- Delete ® *Elimina els punts de ruptura.*

Botons ràpids: Step, Run

Els botons 'Step' i 'Run' realitzen bàsicament les mateixes funcions que Execute - Single Step i Execute - Run, però d'una forma mes abreviada:

Botons de control

1. Pipeline: El botó *Pipeline* mostra una finestra amb totes les etapes de la segmentació del processador DLXV.
2. Clock Cycle Diagram: El botó *Clock Cycle Diagram* mostra una finestra amb un esquema del diagrama de cicles de rellotge del processador DLXV.
3. Virtual FP Stages: Mostra una finestra amb l'estat de les unitats virtuals en coma flotant.
4. Floating Point Stages: Mostra una finestra amb l'estat dels operadors en coma flotant.
5. Vectorial Stages: Esta finestra mostra l'estat de cada una de les unitats d'operació vectorial.
6. Vectorial Lloeu Stages: Esta finestra mostra l'estat de cada una de les unitats de càrrega/emmagatzematge vectorials.
7. Vectorial Registers: Esta finestra mostra informació detallada sobre cada un dels registres vectorials.

Apèndix B: Resum d'instruccions del DLXV

En DLXV l'operació vectorial té el mateix nom que en el DLX, però afegint la lletra "v". Estes són les instruccions vectorials del DLXV. Òbviament DLXV pot executar també les instruccions del DLX.

Instrucció	Funció que realitza
`addv v1, v2, v3`	La màxima elements de V2 i V3, després posa cada resultat en V1.
`addsv v1, f0, v2`	Suma F0 a cada element de V2, després posa cada resultat en V1.
`subv v1, v2, v3`	Resta als element de V2 els de V3 i posa cada resultat en V1
`subvs v1, v2, f0`	Resta a cada element de V2 el valor de F0 i posa cada resultat en V1
`subsv v1, f0, v2`	Resta a F0 cada element de V2 i posa el resultat en V1.
`multv v1, v2, v3`	Multiplica cada element de V2 i V3, després posa els resultats en V1.
`multsv v1, f0, v2`	Multiplica F0 per cada element de V2 i posa cada resultat en V1.
`divv v1, v2, v3`	Dividix cada element de V2 i V3, després posa cada resultat en V1.
`divvs v1, v2, f0`	Dividix cada element de V2 per F0 i posa cada resultat en V1.
`divsv v1, f0, v2`	Dividix F0 per cada element de V2 i posa cada resultat en V1.
`lv v1, r1`	Càrrega V1 des de memòria començant en la direcció R1.
`sv r1, v1`	Emmagatzema V1 en memòria començant en la direcció R1.
`lvws v1, r1, r2`	Càrrega V1 des de memòria començant en la direcció R1 amb stride R2.
`svws r1, r2, v1`	Emmagatzema V1 en memòria començant en la direcció R1 amb stride R2.
`lvi v1, (r1+v2)`	Càrrega V1 des de memòria, des de les direccions R1+V2(i). És a dir, V2 és un vector d'índexs.
`svi (r1+v2), v1`	Emmagatzema V1 en memòria a les direccions R1+V2(i). És a dir, V2 és un vector d'índexs.
`cvi v1, r1`	Crega un vector d'índexs emmagatzemant en V1 els valors: 0, 1·R1, 2·R1, ..., 63·R1
`s_v v1, v2`	Compara (EQ, NE, GT, LT, GE, LI) els elements de V1 i V2. Si la condició és certa posa un 1 en el bit corresponent del vector; en qualsevol cas posa un 0. Posa el vector de bits resultant en el registre de màscara vectorial (vm).
`s_sv f0, v1`	Realitza la mateixa comparació que la instrucció anterior, però utilitzant un escalar (F0) com a operand.
`pop r1, vm`	Compte els uns del registre de màscara vectorial i l'emmagatzema en R1.
`cvm`	Posa tot el registre de màscara vectorial a u.
`movi2s vlr, r1`	Transferix el contingut de R1 al registre de longitud vectorial (vlr).
`movs2i r1, vlr`	Transferix el contingut del registre de longitud vectorial (vlr) a R1.
`movf2s vm, f0`	Transferix el contingut de F0 al registre de màscara vectorial (vm).
`movs2f f0, vm`	Transferix el contingut del registre de màscara vectorial (vm) a F0.
`snesv f0, v1`	Posa vm(i) a 1 si v1(i) ≠ f0