Aufgabe 2.05: Superskalarität

Gegeben ist eine Superskalare Out-of-Order Pipeline, wie sie in der Vorlesung vorgestellt wurde.

a) Was macht der Dispatcher?
Je nach Implementierung ist der Dispatcher für Dispatch alleine oder Dispatch und Issue zuständig. Dispatch ist das Verteilen der Befehle an die verschiedenen Funktionseinheiten. Bei Issue wird ein Befehl zur Ausführung angestoßen. Dabei muss überprüft werden, ob auch alle Operanden eines Befehles zur Verfügung stehen.

b) In welchem Abschnitt der Pipeline findet ein Out-of-Order (OoO) Ausführung und in welchem Abschnitt eine In-Order Ausführung statt?
Die Out-of-Order Ausführung findet in allen Stufen zwischen der oder den Reservation Stations und dem Reorder Buffer statt. Die Befehle werden In-Order in eine Reservation Station geschrieben können aber Out-of-Order gestartet werden. Umgedreht verhält es sich mit dem Reorder Buffer. Die Befehle werden Out-of-Order geschrieben, dürfen Ihn aber nur In-Order verlassen.

c) Welche Aufgaben erfüllt die Completion Unit?
Die Completion-Unit beinhaltet den Reorder Buffer. Sie nimmt die Befehle der Ausführungseinheiten Out-of-Order entgegen und aktualisiert den Zustand im Reorder-Buffer. Sie überprüft, ob der Befehl wegen eines Interrupts, Exception oder spekulative Ausführung verworfen werden muss.

d) Wie werden Pipelinekonflikte durch Namensabhängigkeiten verhindert?
Durch Register Renaming wird verhindert, dass Konflikte durch Namensabhängigkeiten während der Out-of-Order Ausführung entstehen können. Register mit gleichen Namen, die keine Datenabhängigkeiten haben, werden einem anderen internen Register zugewiesen. Das Zurückschreiben der Ergebniswerte (WB oder Retire) findet am Ende wieder In-Order statt und ist somit ebenfalls Namenskonfliktfrei.

e) Wie werden Pipelinekonflikte durch Datenabhängigkeiten verhindert?
Datenabhängigkeiten werden erkannt und beim Laden der Operanden wird nicht der Wert geladen sondern ein Verweis auf ein internes Register vermerkt. Dieses interne Register enthält ein Valid-Bit, das gesetzt wird, sobald der Befehle berechnet ist. In der Reservation Station warten Befehl so lange, bis beide Operanden verfügbar sind. Erst dann wird der Befehl zur Ausführung angestoßen.

f) Warum lohnt sich der ganze Aufwand für eine OOO-Pipeline überhaupt?
Eine superskalare Out-of-Order Pipeline kann gut separate unterschiedliche lange Ausführungspipelines bedienen und dynamisch auf unvorhersehbare Ereignisse reagieren (z.B. Cache Miss beim Speicherzugriff). So können mehrere andere Befehle abgearbeitet werden, während z.B. eine lange Floating-Point Division berechnet wird. Bei einer skalaren In-Order Pipeline müssten Befehle mit kurzen Ausführungszeiten immer auf lange warten.

Aufgabe 2.06: VLIW

Das folgende Codestück soll auf einem VLIW Prozessor mit drei parallelen Ausführungseinheiten (Execution Units) laufen. Geben sie für einen VLIW Prozessor eine möglichst effiziente Befehlsverteilung an, wie sie der Compiler durchführen würde. Die Befehle können beliebig umsortiert werden so lange die Funktionalität identisch bleibt.
Man kann annehmen, dass alle Befehle innerhalb eines Taktes berechnet werden. Ein freies Kästchen ist gleichbedeutend mit einem NOP-Befehl.

```
(1)  add  r1, r2, r3          ; r1 = r2 + r3
(2)  sub  r5, r3, r5          ; r5 = r3 – r5
(3)  ld   r3, [r1]            ; Lade r3 mit [r1]
(4)  mul  r3, r3, r3          ; r3 = r3 * r3
(5)  st   [r5], r3            ; Speichere r3 nach [r5]
(6)  ld   r9, [r7]            ; Lade r9 mit [r7]
(7)  ld   r11, [r12]          ; Lade r11 mit [r12]
(8)  add  r11, r11, r12       ; r11 = r11 + r12
(9)  mul  r11, r11, r9        ; r11 = r11 * r9
(10) st   [r12], r11          ; Speichere r11 nach [r12]
```

a) Nehmen Sie an, dass der Prozessor über drei Ausführungseinheiten verfügt, die jeweils alle Befehle ausführen können.

Slot 1	Slot 2	Slot 3
(1) add r1, r2, r3	(2) sub r5, r3, r5	(7) ld r11, [r12]
(3) ld r3, [r1]	(6) ld r9, [r7]	(8) add r11, r11, r12
(4) mul r3, r3, r3	(9) mul r11, r11, r9	
(5) st [r5], r3	(10) st [r12], r11	

Tabelle 2.2: VLIW Scheduling

b) Nehmen Sie nun an, dass die ersten beiden Ausführungseinheiten arithmetisch-logische Befehle (add, sub, mul) ausführen kann und die letzte für Load/Store Befehle zuständig ist.

Slot 1 (ALU)	Slot 2 (ALU)	Slot 3 (LS)
(1) add r1, r2, r3	(2) sub r5, r3, r5	(7) ld r11, [r12]
	(8) add r11, r11, r12	(3) ld r3, [r1]
(4) mul r3, r3, r3		(6) ld r9, [r7]
(9) mul r11, r11, r9		(5) st [r5], r3
		(10) st [r12], r11

Tabelle 2.3: VLIW Scheduling mit HW Constraints

Aufgabe 2.07: SIMD

a) Zwei MMX (64-Bit SIMD) Register mm0 und mm1 enthalten die folgenden Werte. Tragen Sie das Ergebnis der beiden MMX-Befehle PADDUSB und PADDW in die Tabelle ein.

PADDUSB führt eine unsigned (U) Addition (ADD) mit Saturation (S) auf Packed-Bytes (B) aus.
PADDW führt eine Addition (ADD) auf Packed-Words (W) aus.

Saturation bedeutet, dass bei einem Überlauf z.B. bei einer Addition das Ergebnis auf den höchsten oder niedrigsten Wert gesetzt wird. Ein Unsigned Byte hat z.B. einen Wertebereich von 0x00 – 0xFF. Bei einer 0xA0 + 0xA0 im Zweierkomplement kommt mit Saturation das Ergebnis 0xFF heraus anstatt 0x40.

mm0	10 20 00 80 55 33 FF FF
mm1	01 20 00 80 33 55 00 01
PADDW	11 40 01 00 88 89 00 00
PADDUSB	11 40 00 FF 88 88 FF FF

Tabelle 2.4: SIMD Instruktionen

b) Gegeben ist folgendes Fragment eines C-Programmes:

```
int  i
char a[8],b[8];     // char = 8-Bit Integer
for (i = 0; i < 8; i++) {
     if (a[i] < b[i]) {
          b[i] = i;
     }
}
```

Realisieren Sie den C-Code mittels SIMD Assemblerbefehle. Die Registerbreite der SIMD Befehle beträgt 64-Bit und sie können die Register mittels folgender Befehle als Packed-Byte ansprechen. Die Byte-Order (Little oder Big-Endian) sowie Signed/Unsigned kann vernachlässigt werden. Sie können das Ergebnis wahlweise als Assemblerprogramm oder Datenflussgraphen darstellen.

PMOV mm0, 0x0123456789abcdef	Lädt den hexadezimalen Wert 0x0123456789abcdef in Register mm0
POR mm0, mm1, mm2	Bitweise ODER-Verknüpfung sämtlicher 64-Bit (mm0 = mm1 | mm2)
PAND mm0, mm1, mm2	Bitweise UND-Verknüpfung (mm0 = mm1 & mm2)
PNEG mm0, mm1	Bitweise Negation
PCMPLTB mm0, mm1, mm2	Packed-Byte kleiner Vergleich zwischen mm1 und mm2. Wenn Packed-Byte in mm1 < mm2 ist, wird in mm0 als Ergebnis 0xFF abgelegt, ansonsten 0x00. Beispiel: MM0= 03 71 00 20 01 0A 02 0Fh

	MM1= 04 70 45 67 09 0A 00 04h PCMPLTB MM2, MM0, MM1 MM2= FF 00 FF FF FF 00 00 00h

Tabelle 2.5: SIMD Instruktionen

```
PMOV      mm0, 0x010703090A0B0200      ; a
PMOV      mm1, 0x000305080C0D0405      ; b

PMOV      mm2, 0x0706050403020100
PCMPLTB   mm3, mm0, mm1
PNEG      mm4, mm3
PAND      mm3, mm3, mm2
PAND      mm4, mm4, mm1
POR       mm5, mm3, mm4

Am Ende gilt dann: b = 0x 00 03 05 08 03 02 01 00
```