Der SPARC Prozessor und sein COPRO

Andreas Hennig

17. November 1994

Seminar
der
Projektgruppe Systementwurf
WS94/95 und SS 95

Inhaltsverzeichnis

1 Idee und Geschichte des RISC-Modells

1.1 Geschichtliche Übersicht

Um nicht alle Rechnergenerationen zu wiederholen beginnt dieser Rückblick bei den relativ hoch entwickelten Rechnern der 3. Generation. Diese Rechner verfügen über eine von-Neumann-Architektur (mit Steuerwerk, Rechenwerk, Hauptspeicher, I/O) mit der Fähigkeit zur Nebenläufigkeit (Interruptkonzepte) sowie einen virtuellen Addressraum. Als typisches Beispiel dieser Generation gilt die IBM 360/370. Die Befehlssätze dieser Maschine waren sehr groß und durch hohe Kosten der Microprogrammierung in vier Teile geteilt, die je nach Anwendungsbereich zu dem Rechner hinzugekauft wurden. Man kann also von ersten CISCs (Complex Instruction Set Computer) reden. In der Folgezeit schritt die technische Entwicklung fort und es wurde billiger Microprogramme für Maschinen zu schreiben. Als Folge davon explodierte die Anzahl der Befehle und die Adressierungsarten. Als Beispiel hierfür steht die VAX-Familie (mit über 300 Befehlen). Diese Masse an Befehlen ist bei den folgenden Generationen nicht mehr überschritten worden. Als folgende CISC Rechner kommen dann Prozessoren wie die Intel 80x86 Serie auf, bei denen das Preis/Leistungsverhältnis besser wird, da die Rechner der Vorgeneration doch recht teuer waren. Parallel zu den moderneren CISC-Architekturen kamen dann auch erste RISC (Reduced Instruction Set Computer) auf, die sich durch einen erheblich kleineren Befehlssatz auszeichneten.

1.2 Idee des RISC-Modells

Bei der Beobachtung der CISC-Modelle fällt auf, daß nur ein geringer Teil der Befehle ständig benötigt wird. Alle anderen Befehle treten selten bis nie auf und lassen sich auch teilweise durch andere Befehle umschreiben, was sich oft auch schneller auf die Bearbeitung der Befehle auswirkt. Außerdem wurde es schwer die Instruktionen der CISC-Architekturen auf dem Chip unterzubringen. Die Entwurfsphase dieser Architekturen ist im Gegesatz zu RISC-Arch. wesentlich größer. Auch die Resourcen, die einen Rechner ausmachen, sind unterschiedlich 'wertvoll'. Hieraus resultiert eine Forderung an die Resource Speicher, die mit Hilfe von vielen Registern gut auszunuzten ist. Also Folge davon entstanden circa 1980 erste RISC-Ansätze. Im Folgenden werden RISC und CISC allgemein verglichen. Die wesentlichen Unterschiede zuwischen RISC und CISC sind neben der Anzahl der Befehle:

1. LOAD/STORE-MODELL
2. non-destructive Register
3. konstannte Befehlslänge

zu 1:
bedeutet, daß nur über 'load' und 'store' auf den Speicher zugegriffen werden kann. Alle anderen Operationen und Berechnungen 'laufen' nur auf Registern ab. Auf diese Weise ist es möglich den Lade/Speicher-Aufwand von dem datenbearbeitenden Aufwand (arithmetische und logische Berechnungen) zu trennen. Zusätzlich wird es auch möglich die Befehle effizienter zu verwalten(schedulen), da z.B. ein Ladevorgang 2 bis 3 Prozessorzyklen dauert.
zu 2:
Bei z.B. Additionen auf CISC-Architekturen werden Registerwerte überschrieben, da oft nur wenige Register (meist nur eins) für arithmetische Operationen zur Verfügung stehen. Im Gegensatz hierzu bleiben die Registerwerte bei RISC erhalten, da die Operationen auf den Registern direkt ausgeführt werden.
zu 3:
Durch verschiedene Zugriffsmöglichkeiten auf den Speicher im CISC-Modell, ist es notwendig verschiedene Befehlslängen zu definieren, um den Zugriff zu unterscheiden. Da dieses bei

RISC nicht möglich ist, können alle Befehle die gleiche Länge haben. Als Konsequenz hieraus folgt, daß alle Befehle sich gleich schnell interpretieren lassen und somit eine fast konstante Zeit an Prozessorzyklen benötigen.

2 Der SPARC Prozessor im Überblick

2.1 Die einzelnen Komponenten

Die Abkürzung SPARC steht für Scalable Processor ARChitecture. Der SPARC versteht sich als offene Architektur, d.h. es ist möglich, den Prozessor mit leichten Veränderungen in vielen Problemstellungen einzusetzen. Bei den verschiedenen Einsatzmöglichkeiten bleibt die Binärkompatibilität gewahrt. Scalable bedeutet hier, daß sich der Prozessor leicht an die neuen Umgebungsparameter anpassen läßt. Der Prozessor gehört zur CY7C600 Serie und ist in 25, 33 und 40 Mhz verfügbar.

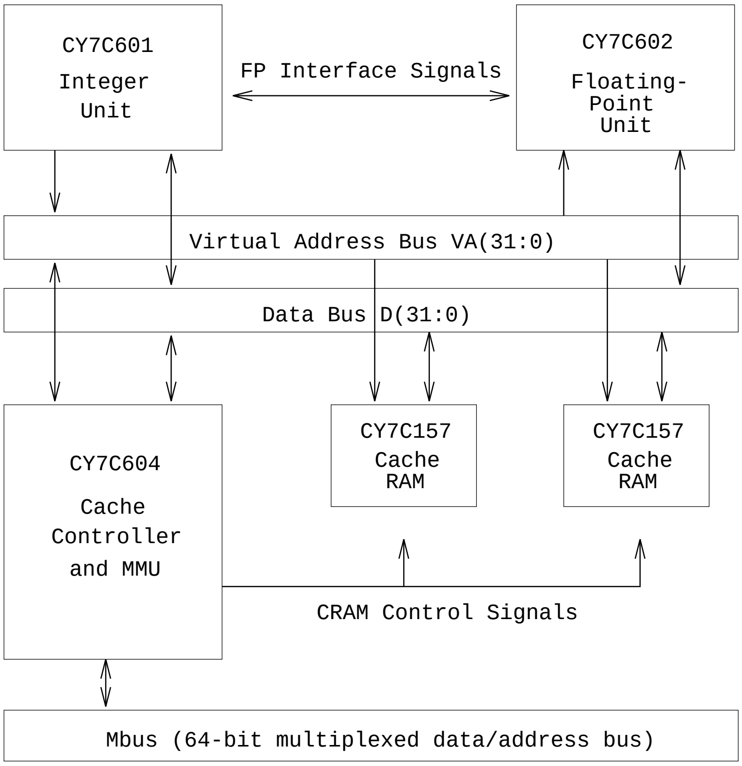

Abbildung 1: Übersicht des SPARC

Das Schema veranschaulicht die einzelnen Komponenten des SPARC. Die Prozessorkomponenten sind:

- Die CY7C601 Interger Unit ist quasi das Herzstück des Prozessors. Sie enthält den Befehlsinterpreter, der alle Befehle, mit Ausnahme der Foatingpoint-Befehle ausführt, oder zumindest die Bearbeitung veranlaßt. Als weitere Merkmale verfügt die I.-Unit über einen Leistungsdurchsatz von im Mittel 1,25 bis 1,5 Taktzyklen pro Befehl, ein großes Registerfile, das in Fenster eingeteilt ist, User- und Supervisormodus sowie Unterstüzung von Semaphoren. Interrups werden innerhalb von 4 bis 7 Taktzyklen, je nach Zustand der Pipelines, bearbeitet.

- Der CY7C602 ist der Coprozessor (FPU) zur I.-Unit. Zusammen mit dem Hauptprozessor ergibt sichder gesamte Befehlssatz des SPARC. Der Coprozessor ist über eine extra

Befehlsleitung mit der I.-Unit verbunden. Beide Prozessoren können parallel arbeiten, wobei aber der Hauptprozessor die Kontrolle behält. Werden Floatingpoint-Befehle gegeben, so werden sie von der 601 addressiert wobei aber beide Prozessoren auf den Datenbus zugreifen. Der Befehl wird auch von Haupt- und Coprozessor übersetzt, wobei aber die 602 erst mit der Berechnung beginnen kann, wenn sie von der 601 das entsprechende Signal erhält. Die Floatpoint-Befehle werden während ihrer Ausführung in einer Warteschlange auf dem Chip der 602 gespeichert, während diese mit den 'normalen' Befehlen fortfährt. Zum Berechnen stehen der FPU 32 32-bit Register zur Verfügung.

- CYC7157 sind Cache-Speicher zur Pufferung von Daten. Der Cache kann sowohl als geregelter Cache (latched), als auch als 'ungeregelter' Cache (write through) eingesetzt werden. Seine Größe beträgt 16K 16-bit Register.

- CYC7604 ist die Memory Management Unit (MMU). Sie verwaltet den Cache-Speicher und bildet den Realspeicher auf den virtuellen Addressraum ab.

2.2 Sonstige Eigenschaften

- Der Registersatz des Prozessors ist in Fenster aufgeteilt. Jedes Fenster besteht aus 24 Arbeitsregistern sowie 8 globalen Registern, die zur Werteübergabe dienen. Alle Register sind 32 Bit breit.

- Der Befehlssatz des Prozessors besteht aus 55 Interger-, 14 Floatingpoint- und 2 Coprozessorbefehlen. Insgesamt lassen sich die Befehle in 5 Kategorien einteilen:

  - Load/Storebefehle zum Zugriff auf den Hauptspeicher
  - Arithmetic/Logical/Shiftbefehle dienen zum eigentlichen Berechnen
  - Control-Transfer Befehle sind Sprungbefehle und Verzeigungen, wobei die daraus resultirenden Konsequenzen für das Pipelining so gut wie möglich berücksichtigt werden.
  - Read/Write Control Register Befehle regeln den Zugriff auf die Register
  - Floating-Point und Coprozessor Befehle fassen alle Befehle zusammen, die die FPU betreffen. Die Befehle für den Coprozessor bestimmt dieser selbst. Die Architektur sieht vor, daß 1024 Befehle für den Coprozessor möglich sind.

3 Die Integer Unit 601 / 611

Da die I.Unit das Herzstück des Prozessors ist, weden im folgenden alle Signale, die die I.U. erreichen bzw. von ihr ausgehen, beschrieben. Die 611 ist eine low cost Version der 601. Die Signalbeschriftungen und Beschreibungen sind identisch. Alle Signale, die mit einem Querbalken über dem Namen markiert sind, sind LOW aktiv; alle anderen sind HIGH aktiv. (HIGH entspricht der logischen 1 und LOW folglich der 0). Der Pfeil gibt an, in welche Richtung die Signale die I.U. passieren.

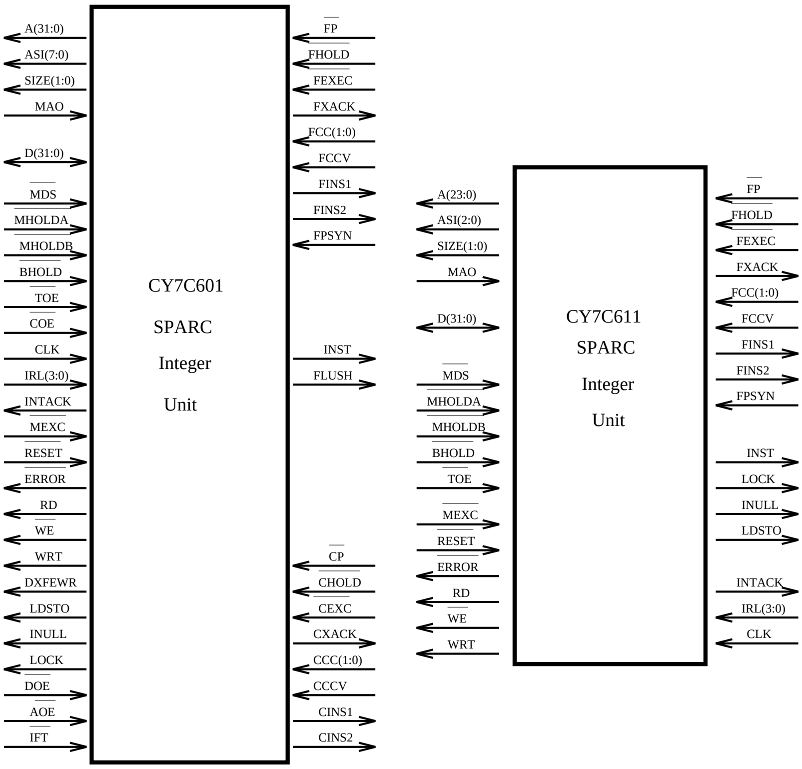

Abbildung 2: Übersicht der externen Signale bei 601/611

3.1 Schnittstelle zur Speicherverwaltung

Die Signale und ihre Bedeutung:

- A(31:0) ist die Leitung auf den Adressbus
- AOE (Adress Output Enable) gibt an, ob auf den A.-Bus geschrieben weden darf
- ASI(7:0) (Adress Space Identifier) gibt an, auf welchen Teil des Speichers sich Befehle beziehen.
- BHOLD (Bus Hold) gibt an, wenn ein externer Bus-Master auf den Bus zugreifen will. Das Setzen dieses Signals hat das Einfrieren der Pipeline zur Folge (bis der Zugriff geklärt ist)
- COE (Control Output Enable) gibt die Signale SIZE, RD, WE, WRT, LOCK, LDSTO und DXFER frei.
- D(31:0) ist der bidirektional Datenbus. (32 Bit breit)
- DOE (Data Output Enable) erlaubt der I.U. auf den Bus Daten zu senden. (Ist nur relevant, wenn ein anderer Bus-Master den Bus kontrolliert)
- DXFER (Data Transfer) gibt an, ob Daten der Befehle über den Bus geladen werden.
- IFT (Instruction cache Flush Trap) gerneriert Trap, falls Signalwert = 0
- INULL (Integer unit NULLify cycle) dient zur Vermeidung von Fehlzugriffen auf den Speicher

- LDSTO (atomic LoaD and STOre) wird gesetzt, falls Lade- oder Speicherbefehle ausgeführt werden.
- LOCK (bus LOCK) wird dann gesetzt, falls die I.U. Kontrolle über den Bus braucht
- MAO (Memory Address Output) ist während eines MHOLD(A/B) gesetzt, um die letzen Zugriffsparameter der Datenbusse zu bekommen.
- MDS (Memory Data Strobe) wird von der Speicherverwaltung gesetzt, um der U.I. mit zuteilen, wann der Datenbus gelesen werden kann.
- MEXC (MEmory EXCeption) wird von der Speicherverwaltung gesetzt, um Fehler im Zugriff auf Speicherinhalte zu melden
- MHOLD(A/B) (Memory Holds) bewirkt, daß beide Clocks (der IU und FPU) eingefroren werden. Dies ist dann notwendig, wenn im Falle eines Cachemiss Daten nachgeladen werden müssen.
- RD (ReaD access) gibt den Status des Zugriffs auf den Datenbus an: RD = 1 steht für lesen und RD = 0 für schreiben.
- SIZE(1:0) (bus transaction SIZE) gibt bei Datenzugriffen an, welcher Teil der Information interpretiert werden muß. Die Tabelle gibt den Type und die Codierung an.

SIZE(1)	SIZE(0)	Daten Transfer Art
0	0	Byte
0	1	Halfword
1	0	Word
1	1	Word (Load/Store Double)

- WE (Write Enable) gibt an, daß Daten zum Schreiben auf dem Bus bereit liegen.
- WRT (advancewd WRiTe) wird gesetzt, um 'schneller' schreiben zu können. (Es ist möglich, um während bestimmter Zyklen, den Bus besser auszulasten)

3.2 Schnittstelle zur FPU / CO-Prozessor

Die Signalleitungen zwischen I.U. und FPU bzw. CO-Prozessor sind keine externen Leitungen. Es handelt sich dabei um quasi interne Leitungen, die zu den Sockeln der CO-Prozessoren führen. Die Schnittstelle(n) sind in ihrem Protokoll und ihrer Arbeitsweise identisch. Es gibt folgende Signale:

- CCC(1:0) (Coprocessor Conditions Codes) gibt Ergebnisse von Vergleichen an die IU weiter.
- CCCV (Coprocessor Conditions Codes Valid) dient erstens zur Synchronisation von IU und CO. Es wird gesetzt, wenn die CCC Leitungen gülitige Werte angenommen haben.
- CXEC (Coprocessor eXCEption) teilt der IU mit, falls es bei der FPU / CO zu Ausnahmen in der Ausführung kommt.
- CHOLD (Coprocessor HOLD) meldet ein Anhalten des CO, was auch ein Halten der IU zur Folge hat. Fortgesetzt wird erst, wenn der CO den Fehler (meistens Zugriffsverzögerung) behoben hat.

- CINS1 (Coprocessor INStruction buffer 1) transportiert das Kontrollsignal an den jeweiligen Coprozessor, sodaß ein gültiger Befehl im Buffer 1 des Befehlsregisters steht. CINS2 dient dem gleichen Zweck. Mit dem dort stehenden Befehl wird gleich (im selben Zyklus) die Auswertung begonnen. Es werden nie beide Leitungen im gleichen Zyklusdurchlauf auf HIGH gelegt.
- CP (Coprocessor Present) gibt an, ob ein CO vorhanden ist.
- CXACK (Coporcessor EXception ACKnowledge) meldet dem CO, daß ein Ausnahmezustand bei dem CO erkannt ist. Dieses Signal hat ein Halten der CO - Pipeline zur Folge.
- FCC(1:0) (Floatingpoint Conditions Codes) gibt Ergebnisse von Vergleichen an die IU weiter.

FCC(1)	CFF(0)	CONDTION
0	0	gleich
0	1	$Op1 < Op2$
1	0	$Op1 > Op2$
1	1	ungeordnet

- FCCV (Floatingpoint Conditions Codes Valid) dient erstens zur Synchronisation von IU und FPU. Es wird gesetzt, wenn die FCC Leitungen gültige Wert angenommen haben.
- FXEC (Floatingpoint eXCEption) teilt der IU mit, falls es bei der FPU zu Ausnahmen in der Ausführung kommt.
- FHOLD (Floatingpoint HOLD) meldet ein Anhalten der FPU, was auch ein Halten der IU zur Folge hat. Fortgesetzt wird erst, wenn die FPU den Fehler (meistens Zugriffsverzögerung) behoben hat.
- FINS1 (Floatingpoint INStruction buffer 1) transportiert das Kontrollsignal an den jeweiligen Coprozessor, sodaß ein gültiger Befehl im Buffer 1 des Befehlsregisters steht. CINS2 dient dem gleichen Zweck. Mit dem dort stehenden Befehl wird gleich (im selben Zyklus) die Auswertung begonnen. Es werden nie beide Leitungen im gleichen Zyklusdurchlauf auf HIGH gelegt.
- FP (Flaotingpoint Present) gibt an, ob eine FPU vorhanden ist.
- FXACK (Flaotingpoint EXception ACKnowledge) meldet der FPU, daß ein Ausnahmezustand bei der FPU erkannt ist. Dieses Signal hat ein Halten der FPU - Pipeline zur Folge.
- FLUSH (floatingpoint / coprocessor instruction FLUSH) veranlaßt die FPU / CO ihren aktuellen Befehl in den Instruction-Buffer zu sichern. Das Signal wird gesetzt, wenn die IU ein Trap-Signal erhält.
- INST (INSTruction fetch) signalisert der FPU / dem CO, daß ein neuer Befehl 'geholt' wird. Zu diesem Zeitpunkt wird der neue Befehl in eine der Instruction-Buffer geladen.

3.3 Interrupt und Kontrollsignale

Diese Klasse von Signalen beschäftigt sich mit der Kontrolle der IU und externen Ereignissen.

- ERROR (ERROR state) wird gesetzt, wenn die IU in den Fehlerzustand geht. Zuvor werden Porgramm Counter (PC) und einige Statusregister gesichert. Die einzige Möglichkeit aus diesem Status heraus zu kommen, ist ein zeitgesteuertes Setzen der RESET-Leitung.
- INTACK (INTerrupt ACKnowledge) meldet der IU das Auftreten eines Interrups. Aber nur falls diese nicht ausmaskiert oder auf 'hängend' geschaltet sind.

3.4 Einige Signallauf Diagramme

Im Folgenden sind einige Signalläufe der E/A-Signale dargestellt.

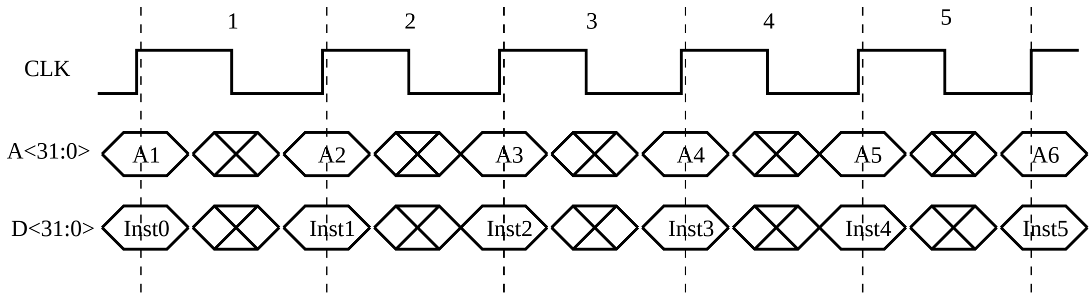

Abbildung 3: Instruction Fetch

In dem Diagramm wird das Laden von Befehlen dargestellt. Man kann sehr gut sehen, wie die Bearbeitungszyklen des Ladens funktionieren: zuerst wird die Addresse des nächsten Befehls geladen. Im folgenden Zyklus liegt der Befehl dann am Datenbus an.

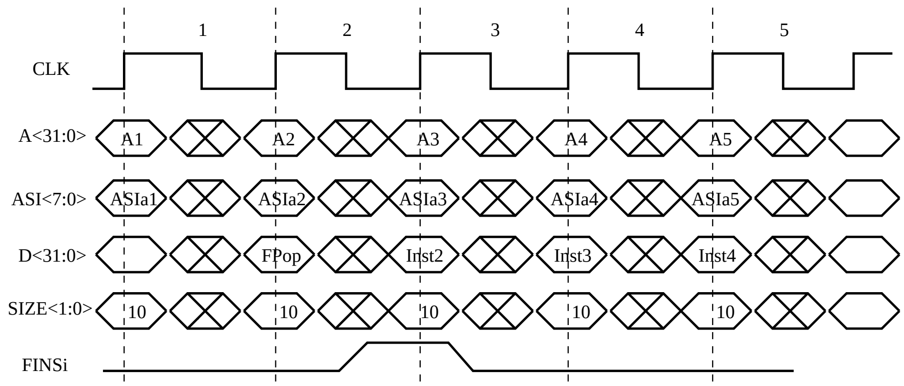

Abbildung 4: Floatingpoint Operation Timing

Im Wesentlichen gibt es für die IU keinen Unterschied im Signalverhalten bei Floatbefehlen. Es muß lediglich noch das FINS1/2 - Signal gesetzt werden, damit die FPU den Befehl mit bearbeitet. Im Beispiel ist die erste Anweisung ein Float-Befehl.

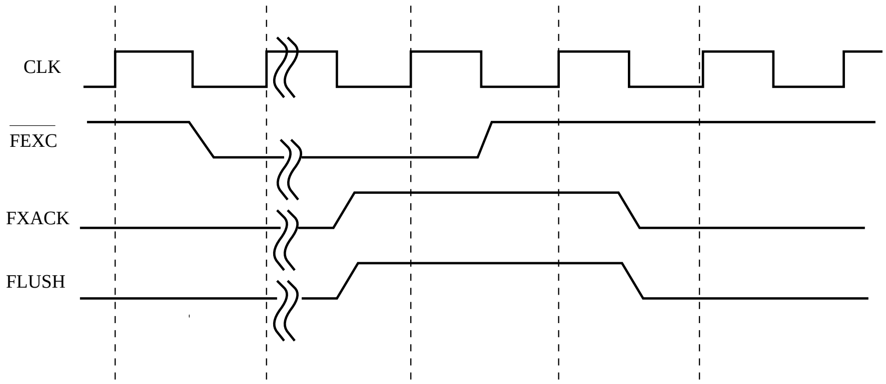

Abbildung 5: Floatingpoint Exception

Tritt bei der FPU ein Ausnahmezustand auf, so wird an die IU ein FEXC - Signal gesendet. Daraufhin veranlaßt die IU die FPU ihren Befehlsbuffer zu löschen (FLUSH) und löst einen Interrupt aus, der die FPU zur 'Lösung' des Problems veranlaßt (FXACK). Erst wenn das Problem, welches zum Ausnahmezustand führte, gelöst ist gibt die Interrupt Service Routine die Kontrolle wieder zurück.

3.5 Das Befehlsformat

Die Befehle sind immer 32 Bit lang. Es kann vorkommen, daß je nach Befehl nur Teile von den 32 Bit benötigt werden. Die Tabelle gibt hier eine Übersicht über die FPU / Co-Befehle.

Mnemonic	31 30	29 ... 25	24 ... 19	18 ... 14	13 ... 5	4 ... 0
CPOP1	1 0	rd	110110	rs1	OPC	rs2
CPOP2	1 0	rd	110111	rs1	OPC	rs2
FABSs	1 0	rd	110100	ignored	0 0 0 0 0 1 0 0 1	rs2
FADDs	1 0	rd	110100	rs1	0 0 1 0 0 0 0 0 1	rs2
FADDd	1 0	rd	110100	rs1	0 0 1 0 0 0 0 1 0	rs2
FADDx	1 0	rd	110100	rs1	0 0 1 0 0 0 0 1 1	rs2
FCMPs	1 0	ignored	110101	rs1	0 0 1 0 1 0 0 0 1	rs2
FCMPd	1 0	ignored	110101	rs1	0 0 1 0 1 0 0 1 0	rs2
FCMPx	1 0	ignored	110101	rs1	0 0 1 0 1 0 0 1 1	rs2
FCMPEs	1 0	ignored	110101	rs1	0 0 1 0 1 0 1 0 1	rs2
FCMPEd	1 0	ignored	110101	rs1	0 0 1 0 1 0 1 1 0	rs2
FCMPEx	1 0	ignored	110101	rs1	0 0 1 0 1 0 1 1 1	rs2
FDIVs	1 0	rd	110100	rs1	0 0 1 0 0 1 1 0 1	rs2
FDIVd	1 0	rd	110100	rs1	0 0 1 0 0 1 1 1 0	rs2
FDIVx	1 0	rd	110100	rs1	0 0 1 0 0 1 1 1 1	rs2
FMOVs	1 0	rd	110100	ignored	0 0 0 0 0 0 0 0 1	rs2
FMULs	1 0	rd	110100	rs1	0 0 1 0 0 1 0 0 1	rs2
FMULd	1 0	rd	110100	rs1	0 0 1 0 0 1 0 1 0	rs2
FMULx	1 0	rd	110100	rs1	0 0 1 0 0 1 0 1 1	rs2
FNEGs	1 0	rd	110100	ignored	0 0 0 0 0 0 1 0 1	rs2

Fortsetzung der Tabelle:

Mnemonic	31 30	29 ... 25	24 ... 19	18 ... 14	13 ... 5	4 ... 0
FSQRTs	1 0	rd	110100	ignored	0 0 0 1 0 1 0 0 1	rs2
FSQRTd	1 0	rd	110100	ignored	0 0 0 1 0 1 0 1 0	rs2
FSQRTx	1 0	rd	110100	ignored	0 0 0 1 0 1 0 1 1	rs2
FSUBs	1 0	rd	110100	rs1	0 0 1 0 0 0 1 0 1	rs2
FSUBd	1 0	rd	110100	rs1	0 0 1 0 0 0 1 1 0	rs2
FSUBx	1 0	rd	110100	rs1	0 0 1 0 0 0 1 1 1	rs2
FdTOi	1 0	rd	110100	ignored	0 1 1 0 1 0 0 1 0	rs2
FdTOs	1 0	rd	110100	ignored	0 1 1 0 0 0 1 1 0	rs2
FdTOx	1 0	rd	110100	ignored	0 1 1 0 0 1 1 1 0	rs2
FiTOd	1 0	rd	110100	ignored	0 1 1 0 0 1 0 0 0	rs2
FiTOs	1 0	rd	110100	ignored	0 1 1 0 0 0 1 0 0	rs2
FiTOx	1 0	rd	110100	ignored	0 1 1 0 0 1 1 0 0	rs2
FsTOd	1 0	rd	110100	ignored	0 1 1 0 0 1 0 0 1	rs2
FsTOi	1 0	rd	110100	ignored	0 1 1 0 1 0 0 0 1	rs2
FsTOx	1 0	rd	110100	ignored	0 1 1 0 0 1 1 0 1	rs2
FxTOi	1 0	rd	110100	ignored	0 1 1 0 1 0 0 1 1	rs2
FxTOs	1 0	rd	110100	ignored	0 1 1 0 0 0 1 1 1	rs2
FxTOd	1 0	rd	110100	ignored	0 1 1 0 0 1 0 1 1	rs2

4 Die Anforderungen an einen Coprozessor

Das Prozessorsystem stellt einen zusätzlichen Sockel für den Anschluß eines selbstdefinierten Coprozessors bereit. Um diesen zu nutzen, ist es jedoch notwendig, daß dieser bestimmte Kriterien erfüllt. Die Signalleitungen sind in dem Kapitel der IU-Beschreibung beschrieben. Das Protokoll dieser Leitungen ist mit dem der FPU identisch. Das genaue Protokoll wird im folgenden Kapitel beschrieben. Es gibt im Wesentlichen vier Anforderungen, die an einen selbstdefinierten Coprozessor gestellt werden:

- das Registermodell muß eingehalten werden
- es muß ein Statusregister geben
- es müssen Befehlswarteschlangen vorhanden sein
- es muß pinkompatibel sein (logisch)

Da die IU für die eigentliche Steuerung verantwortlich ist, kann ein Coprozessor selbst keine Daten adressieren, d.h. alle Adressierungen laufen über die IU. Hierzu ist es notwendig, daß der Coprozessor auf den gleichen Bus zugreift wie die IU. Die Coprozessoren arbeiten wie die IU nur auf Registern. Hierzu hat ein CO ein 32 x 32 Bit breites Registerfile. Alle CO-Befehle arbeiten auf diesen Registern. Sollen also Daten in die Register geladen oder aus den Registern geschrieben werden, so geschieht dieses über den Bus, auf den vorher, von der IU veranlaßt, die Daten gelegt wurden. Auch das Statusregister des CO kann nur auf diese Weise ausgelesen werden. Ein Coprozessor muß auch über zwei Address- und Instruction-Decode-Register verfügen. Alle Befehle werden simultan zur IU gelesen und interpretiert, folglich ist die Decode- Einheit identisch. Das CFlush-Signal, das von der IU gesetzt werden kann löscht diese beiden Register, läßt aber die Befehlswarteschlange unberührt. In die Warteschlangen werden Befehle eingelagert, die aufgrund von Ausnahmen nicht sofort bearbeitet werden können. (Näheres hierzu in der Beschreibung des Exceptionmodells.) Die beiden Warteschlangen werden nach dem FIFO - Prinzip verwaltet.

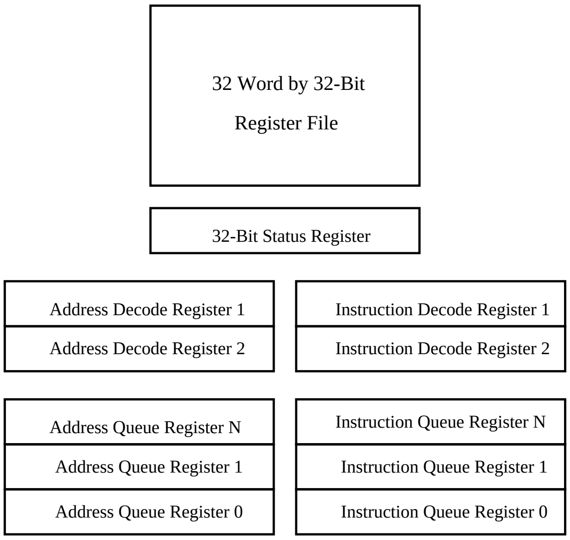

Abbildung 6: Registermodell des Coprozessors

5 Die FPU 602 als Beispiel für einen Copozessor

Die FPU 602 ist der Standard - Coprozessor zur Sparc IU. Er verfügt über eine 64-Bit ALU (Arithmetic and Logical Unit), mit deren Hilfe 64-Bit Additionen, Subtraktioen, Multipikationen, Divisoinen und Wurzelziehen möglich ist. Zu den 64-Bit Operationen müssen allerdings zwei 32-Bit Register zusammengenommen werden. Die FPU verfügt, gemäß den Anforderungen an einen CO, über 32 dieser 32-Bit Register. Verarbeitet werden Gleitkommazahlen gemäß des ANSI/IEEE-754 Standards. Wird die FPU an 40 MHz angeschlossen, so kommt man auf einen Leistung von 6,15 MFLOPS (Million-FloatingPoint-OperationsPerSecond) bei Verwendung von doubble-precision Zahlen.

5.1 Die Schnittstelle der FPU

Die FPU hat ein Interface, das genau der Spezifikation für Coprozessoren folgt. Alle Signalleitungen, die zur IU führen, sind in dem Kapitel der IU beschrieben. Zusätzlich bestehen noch Verbindungen zur MMU 604 / 605:

- DOE (input, Data Output Enable) wird gesetzt, wenn die Daten wieder auf den Bus gelegt werden dürfen.
- MHOLD A/B (input, Memory HOLD) wird gesetzt, wenn die Pipeline angehalten werden muß, weil ein Cachemiss aufgetreten ist. (analog zu den HOLD A/B Leitungen der IU)
- BHOLD (input, Bus HOLD) ist gesetzt, wenn ein externer Busmaster den Bus kontrolliert. Während dieser Zeit ist die Pipeline angehalten.
- MDS (input, Memory Data Strobe) wird gesetzt um Daten in die FPU zu laden, wenn die FPU-Pipline angehalten ist. (MHOLD A/B oder BHOLD)

- FNULL (output, Fpu NULLify cycle) hat die gleiche Funktion wie das INULL-Signal.
- RESET (input, RESET) wird gesetzt um die FPU zu (re)initialisieren. Das Signal bewirkt ein Setzen aller schreibbaren Elementen auf Null. Das Signal muß mindesten 8 Taktzyklen anliegen.
- CLK (input, CLocK) ist das Taktsignal, das die FPU steuert.

Ansonsten bestehen noch Leitungen zu dem Adressbus A(0:31) und zum Datenbus D(0:31). Das folgende Schema stellt die FPU mit den Signalleitungen dar.

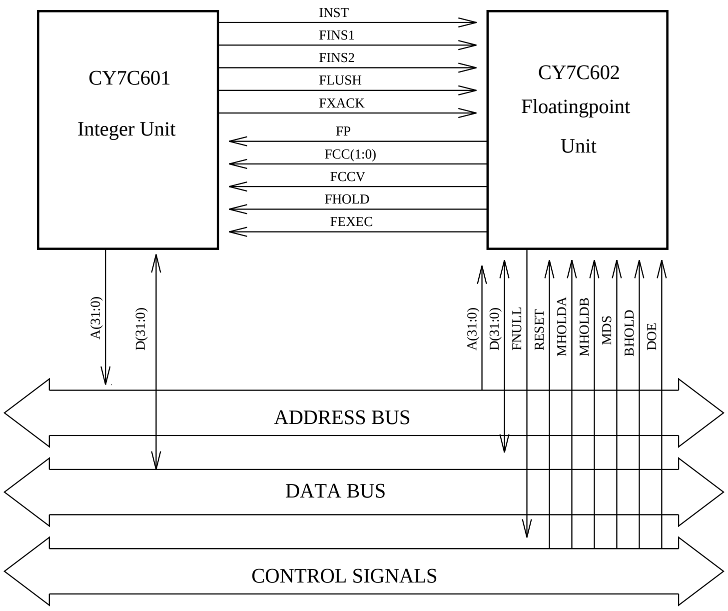

Abbildung 7: Interface der FPU

5.2 Blockschema der FPU

Die FPU läßt sich in 9 funktionale Teilblöcke teilen, von denen jeder eine bestimmte Aufgabe erfüllt.

1. die Fetch Unit greift Daten und Adressen von den Bussen ab.
2. die Load Unit übernimmt die Daten und Adressen von der Fetch Unit und 'merkt' sich diese, bis sie in ein Register gespeichert werden.
3. die Decode Unit entschlüsselt die Befehlscodes, die von Bus gelesen werden.
4. die Exectue Unit enthält die Befehls- und Adressqueue. Sie führt die Befehle aus, indem von hieraus die entsprechenden Multiplexer und die ALU gesteuert werden.
5. die Store Unit 'hält' Daten, die auf den Bus zurück geschrieben werden sollen bis der Bus frei ist und die Daten übertragen werden können.
6. die Exception Unit regelt das Fehlerprotokoll mit der IU, falls sie von der Execute Unit dazu veranlaßt wird.
7. das 32-Bit Registerfeld stellt 32 32-Bit Register zur Verfügung.
8. die ALU führt die eigentlichen Berechnungen durch.

9. die Dependency checking Unit prüft ob ein Nullzyklus vorliegt, ob also Daten für einen Berechnungszyklus fehlen. Im Bedarfsfall wird dann das FNULL-Signal gesetzt.

Das folgende Übersichtsschema stellt die Verbindungen der einzelnen Kompomenten untereinander und mit den anderen Prozessorteilen dar.

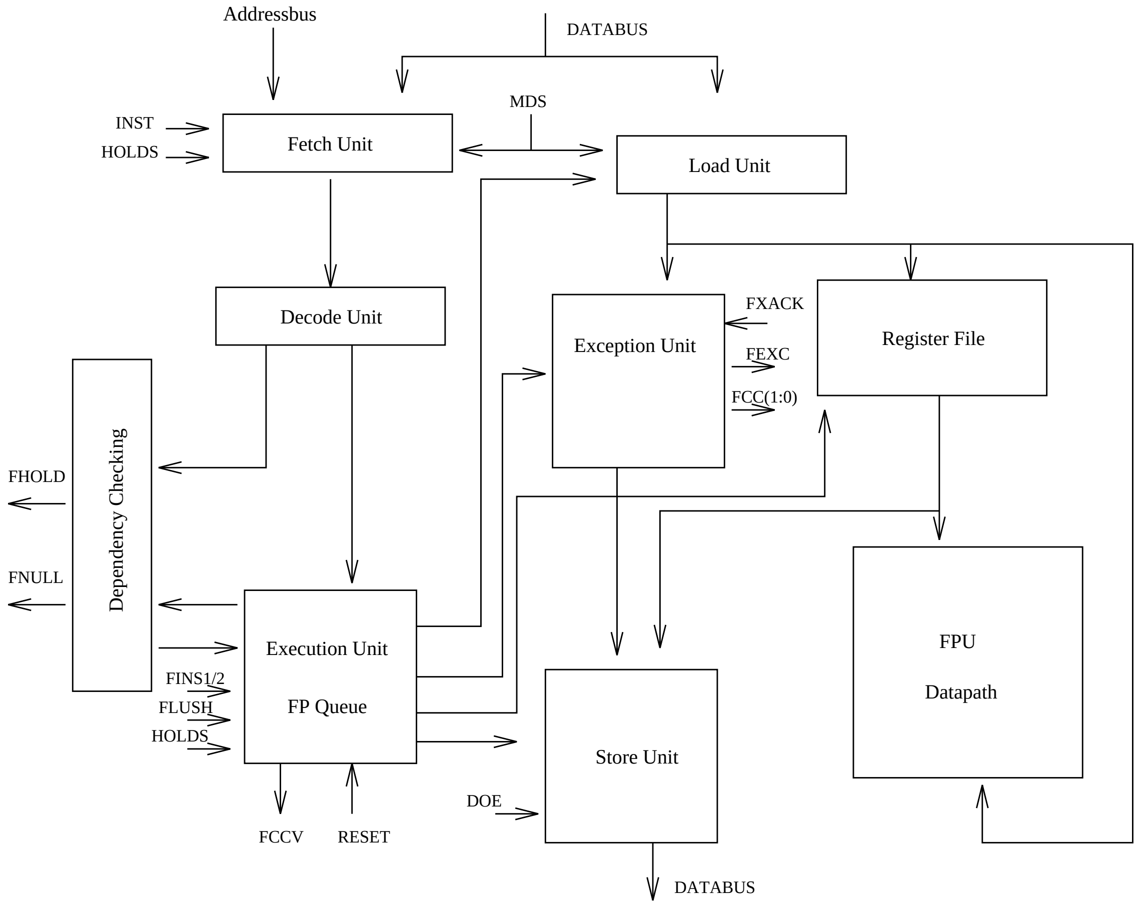

Abbildung 8: Blockdiagramm der FPU

5.3 Das Fetch - Execution Modell der FPU

Die FPU benutzt eine vier Zustände Pipeline. Die Zustände sind fetch, decode, execute und write (F, D, E, W). Da die beiden Prozessoren des SPARC kooperativ arbeiten und es Befehle bei den Floatbefehlen gibt, die länger dauern als einen Bearbeitungszyklus, verweilt die FPU im W-State, falls mehrere Zyklen erforderlich sind. Aufgrund der LOAD / STORE - Architektur gibt es drei Arten von Befehlen, die hier unterschieden werden können.

- LOAD Befehle
- STORE Befehle
- Float Befehle

Die folgenden Tabellen stellen die Aktionen dar, die während dieser Zyklen ablaufen.

Ausführen eines LOAD-Befehls

Zyklus	Handlung
D-Zustand	Decodieren und Prüfen, ob Abhängigheiten bestehen
E-Zustand	FHOLD setzen falls nötig
W-Zustand	Abgreifen der Daten vom Bus
Wh1-Zustand	Setzen des FSR
Wh2-Zustand	Schreiben der Daten in die Register

Ausführen eines STORE-Befehls

Zyklus	Handlung
D-Zustand	Decodieren und Prüfen ob Abhängigheiten bestehen
E-Zustand	FHOLD setzen falls nötig, Daten aus Registern lesen
W-Zustand	Daten auf den Bus legen
Wh1-Zustand	Daten vom Bus nehmen
Wh2-Zustand	Bus freigeben

Ausführen eines FLOAT-Befehls

Zyklus	Handlung
D-Zustand	Übersetzen des Befehls, prüfen ob Resourcen verfügbar
E-Zustand	FHOLD setzen falls nötig, Daten aus Registern lesen
W-Zustand	Lesen noch benötigter Daten, Berechnen
FP Queue	Berechnen der FOPs in der Schlange
.	.
.	.
.	.
FP Queue	Prüfe auf Sonderfälle
FP Queue	Aktualisiere FSR

5.4 Das Register - Modell der FPU

Das Registerfeld der 32 32-Bit Register muß je nach berechnender Zahl umstrukturiert werden. Anschaulich lassen sich die Register wie folgt anordnen:

f0	f1	f2	f3
f4	f5	f6	f7
f8	f9	f10	f11
f12	f13	f14	f15
f16	f17	f18	f19
f20	f21	f22	f23
f24	f25	f26	f27
f28	f29	f30	f31

Hierbei enthält jedes Register einen single precision Wert oder einen signed interger Wert. Benutzt man nun double precision Werte, werden einfach benachbarte Register zu einem verschmolzen. Dabei sind dann einfach nur alle geraden Adressen zu nutzen:

f0	f1	f2	f3
MSW	LSW	MSW	LSW

Erweitert man die Genauigkeit der Zahlen auf 128 Bit, so lassen sich nur noch 8 Werte dieser Genauigkeitsstufe speichern. Adressiert werden dann nur noch die Adressen der ganzzahligen Vielfachen von 4 (Allerdings nur bis 28: 0,4,8,12,16,20,24,28).

f0	f1	f2	f3
MSW			LSW

Als Folge der Reduzierung der Anzahl der Register bei der Zusammenlegung werden bei der single Darstellung noch alle 5 Bits benötigt. Bei der double Darstellung wird das LSB ignoriert. In der extended Darstellung entfallen dann die letzen beiden LSBs.

Variablen Type	4	3	2	1	0
single					
double					X
extended				X	X

5.5 Das Floatingpoint-Unit Status Register FSR

Um den Zustand der FPU auslesen zukönnen werden alle wichtigen Angaben über den aktuellen Zustand in einem 32-Bit breiten Register, dem FSR, gesichert. Diese Einstellungen werden dann im Bedarfsfall über den Bus ausgelesen. Das folgende Schema und die Tabelle stellen die genaue Bedeutung der einzelnen Bits dar.

RD	RP	TEM	NS	R	version	FTT	QNE	R	FCC	AEXC	CEXC
31 30	29 28	27 ... 23	22	21 20	19 ... 17	16 ... 14	13	12	11 10	9 ... 5	4 ... 0

Dabei gliedern sich die Bereiche TEM, AEXC und CEXC nochmals in Unterpunkte, die in der folgenden Tabelle näher erklärt werden.

TEM:

nvm	ofm	ufm	dzm	nxm

AEXC:

nva	ofa	ufa	dza	nxa

CEXC:

nvc	ofc	ufc	dzc	nxc

Feld	Werte	FSR Bits	Beschreibung	ladbar
RD	0 - runde auf nächstes 1 - runde auf 0 2 - runde auf $+\infty$ 3 - runde auf $-\infty$	31:30	Rundungsrichtung	ja
RP	0 - Extended precision 1 - Single precision 2 - Double precision 3 - reserviert	29:28	extended Rundungsgenauigkeit	ja
TEM	0 - ausschalten des IRQ 1 - einschalten des IRQ NVM OFM UFM DZM NXM	27:23 27 26 25 24 23	IRQ Einschalt-Maske ungültige IRQ-Maske Überlauf der Maske Unterlauf der Maske Teilung durch Null Maske ungenaue IRQ-Maske	ja
NS	 0 - eingeschaltet 1 - ausgeschaltet	22	NICHT STANDARD Floatingpoint 0 = IEEE-Modus 1 = schnelle Berechnung	ja
version	0 - 7	19:17	FPU Versionsnummer	nein
FIT	0 - keiner 1 - IEEE-Exception 2 - nicht beendete FOP 3 - nicht implermentierte FOP 4 - Sequenz Fehler 5 - 7 Reserviert	16:14	Art des IRQs	nein
QNE	0 - Schlange nicht leer	13	FPU-Queue nicht leer	nein
FCC	0 - = 1 - < 2 - > 3 - ungeordnet	11:10	FPU Vergleichsergebnisse	ja
AEXC	 NVA OFA UFA DXA NXA	9:5 9 8 7 6 5	Aufgetretene Exception Bits Ungültigkeits-Exception Überlaufs-Exception. Unterlaufs-Exception Teilung durch Null-Exception ungenaue exc.	ja
CEXC	 NVA OFA UFA DXA NXA	4:0 9 8 7 6 5	Aktuelle Exception Bits Ungültigkeits-Exception Überlaufs-Exception Unterlaufs-Exception Teilung durch Null-Exception ungenaue exc.	ja
R	immer auf 0 gesetzt	21,20,12	reservierte Bits	nein

5.6 Der Befehlssatz der FPU

Generell kann man drei Arten von Befehlen hier unterscheiden. Als erstes gibt es die Gruppe der LOAD / STORE - Befehle. Sie sind, wie der Name schon sagt, für den Transfer von Daten zwischen Speicher und Registern zuständig.

- LDF und LDDF laden Daten aus dem Speicher in die Register.
- STF und STDF schreiben Daten aus Registern in den Speicher zurück.
- LDFSR und STFSR lesen das FSR aus bzw. (über)schreiben es neu.
- STDFQ erlaubt es, die FP-Queue auszulesen. (Es handelt sich hierbei um eine previligierte Operation.)

Als zweite Gruppe gibt es die Floatingpoint-Vergleichsbefehle. Sie vergleichen zwei Operatoren miteinander. Da es mehrere Typen von Floatingpointwerten gibt, existiert für jeden Type ein extra Vergleichsbefehl. Dabei gibt es noch die Möglichkeit, einen Interrupt auszulösen, falls kein Vergleich der Zahlen zulässig ist.

- FCMPs vergleicht Single-Werte.
- FCMPd vergleicht Double-Werte.
- FCMPEs vergleicht Single-Werte und löst einen Interrupt aus, falls Vergleich nicht zulässig war.
- FCMPEd vergleicht Double-Werte und löst einen Interrupt aus, falls Vergleich nicht zulässig war.

Zuletzt bleiben noch die eigentlichen, arithmetischen Floatingpointoperatoren, die Daten in den Registern verändern. Man kann auch diese große Gruppe nochmals in den Teil der mathematischen Operatorenund den Teil der konvertierenden Operatoren gliedern. Dabei ist auch bei jedem Befehl definiert, auf welche Daten er sich bezieht. Als mathematische Operatoren gibt es:

- FABSs bildet den Absolutwert einer Zahl.
- FADDs addiert Single-Werte.
- FADDd addiert Double-Werte.
- FDIVs dividiert Single-Werte.
- FDIVd dividiert Double-Werte.
- FMULs multipliziert Single-Werte.
- FMULd multipliziert Double-Werte.
- FSUBs subtrahiert Single-Werte.
- FSUBd subtrahiert Double-Werte.
- FSQRTs zieht Quadratwurzel eines Single-Wertes.

- FSQRTd zieht Quadratwurzel eines Double-Wertes.
- FMOVs verschiebt Werte in Registern.

Die konvertierenden Funktionen ermöglichen es jede Zahl in eine andere Darstellung umzuwandeln. Dabei kann es allerdings zu Verlusten kommen. Diese Verluste müssen vom Anwender bedacht werden. Jeder Befehl dieser Gruppe hat die Form: FxTOy ,wobei x und y aus der Menge i, s, d sind. 'i' steht dabei für Integerzahl, 's' und 'd' jeweils für Single und Double.

5.7 Das Exception-Modell

Kommt es bei der Berechnung innerhalb der FPU zu Fehlern, sei es duch die Operanden, z.B. Teilen durch Null, oder weil es gewünscht ist (Fehler bei Unvergleichbarkeit), so setzt die FPU das FEXC-Signal. Die I.U. erkennt dann den 'Hilferuf' der FPU und setzt dann das FACK-Signal, welches der FPU signalisiert, daß der Fehlerfall bemerkt worden ist. Gleichzeitig wird noch das FLUSH-Signal mit gesetzt. Dieses veranlaßt die FPU ihre beiden Adress- und Instructiondecode-Register zu löschen. Sind die Register gelöscht, so werden die Befehle abgearbeitet, die vor dem Fehlerfall in Bearbeitung waren. Da kurz nach dem Fehler noch zu bearbeitende Befehle in die Warteschlangen geschrieben werden, können sie dann bearbeitet werden. Die folgende Skizze veranschaulicht die drei Zustände, die das Exception-Modell hat.

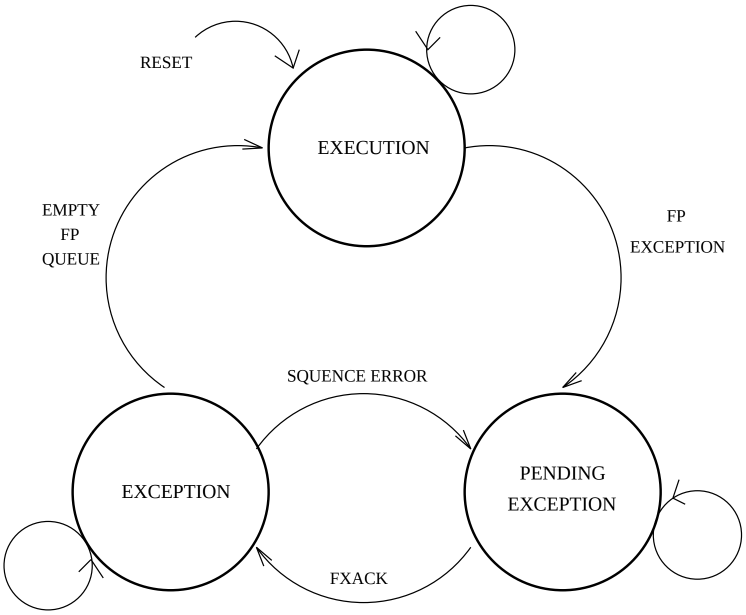

Abbildung 9: Exception-Modell

Literatur

[1] Ross Technology, Inc. *SPARC RISC USER'S GUIDE*