Atari SFP-004 an einem Beispiel

Atari SFP-004 ist eine FPU-Erweiterung für den Mega ST. Die Integration der optionalen FPU im Mega STE ist technisch äquivalent. Es existiert sehr wenig Dokumentation zur Programmierung der FPU im SFP-004, weshalb im Folgenden die Programmierung an einem Beispiel vorgestellt werden soll. Das Beispiel ersetzt keine Dokumentation der FPU. Hierzu sei auf folgende Literatur verwiesen, die auszugsweise auch in diesem Beispiel verwendet wird: [1], [2].

UNTERSCHIEDE ZUM 68020/68030

Es ist wichtig, sich zunächst den Unterschied zur FPU-Integration in einem 68020/68030-basierten System, wie z. B. dem Atari TT, vor Augen zu führen. Dort verwendet der Programmierer eine FPU-Instruktion im Code, einen Line-F-Opcode, dessen ersten Nibble $F ist. Die CPU und die FPU kommunizieren dann über den Bus miteinander, um die mathematische Operation, einen eventuellen Datentransfer, Fehlerbehandlung usw. abzustimmen. Dies läuft alles für den Programmierer unsichtbar ab.

Der 68000 in einem Mega ST/STE besitzt jedoch kein Coprozessor-Interface. Aus diesem Grund muss das – beim 68020/68030 in Hardware integrierte – Interface hier in Software, d. h. durch den Programmierer, emuliert werden.

DAS INTERFACE DES SFP-004

Das Businterface der FPU wird in Form diverser Register in den Adressraum zwischen $FFFA40 und $FFFA60 eingeblendet. Details sind in [2] beschrieben. Die wichtigsten Register sind:

$FFFA40: Response, 16 Bit: Zustand und Status der FPU können hier ausgelesen werden.
$FFFA4A: Command, 16 Bit: Der Befehl an die FPU wird an diese Adresse geschrieben.
$FFFA50: Operand, 32 Bit: Transfer von Operanden (d. h. Daten) zwischen CPU und FPU.

EIN BEISPIEL

Die Programmierung wird im Folgenden am – willkürlich gewählten – Beispiel der Implementierung der Arcuscosinus-Funktion für den SFP-004 [3] erklärt. Es wird sowohl der Assembler-Quelltext gezeigt als auch eine interaktive Ausführung der einzelnen Schritte in GFA Basic.

```
lea     0xfffa50,%a0
movew   #0x541c,%a0@(comm)
```

```
OK >sdpoke $FFFA4A,$541C
```

Laden des Befehls ins Command-Register. Der FPU-Befehl ist immer im *zweiten* Wort des entsprechenden Line-F-Opcodes codiert.[1] Daher kann [4] verwendet werden, um den Befehl zu decodieren:

FACOS **Arc Cosine (MC6888X, M68040FPSP)** **FACOS**

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	1	1	0	0

[1] Das erste Wort der Opcodes enthält Informationen, die für die CPU (68020/68030) wichtig sind, z. B. woher/wohin eventuelle Datentransfers erfolgen sollen, also die *effective address.*

Konkret handelt es sich beim Befehl $541C folglich um den Arcuscosinus, wobei die Quelle Daten von der CPU sind (R/M = 1), die im Double-Precision-Real-Format vorliegen (Source Specifier = 101). Das Ergebnis soll im FPU-Register FP0 abgelegt werden (Destination Register = 000).

```
cmpiw   #0x8900,%a0@(resp)          OK >print hex$(dpeek($FFFA40))
                                    1608
```

Die FPU beginnt mit der Abarbeitung eines Befehls frühestens nach dem erstmaligen Lesen des Response-Registers. Obwohl es sich bei dem Befehl im Beispiel um einen Vergleich handelt (CMPI), wird das Ergebnis des Vergleichs nicht genutzt. So würde z. B. eine eventuelle Fehlermeldung der FPU durch diesen Code nicht abgefangen. Der Code geht davon aus, die erwartete Antwort zu erhalten, in diesem Fall $1608, wobei es sich gemäß [1] um die *Evaluate Effective Address and Transfer Data Primitive* handelt:

7.4.2.2 EVALUATE EFFECTIVE ADDRESS AND TRANSFER DATA PRIMITIVE. This primitive is used by the FPCP to request the transfer of a data item between the floating-point data and control registers and an external location (either memory or a main processor register). The format of this primitive is shown in Figure 7-9. The main processor services this request by evaluating the effective address indicated by the F-line word of the instruction and transferring the number of bytes indicated by the length field of the primitive to or from the operand CIR.

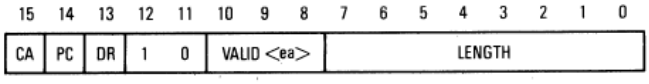


Figure 7-9. Evaluate Effective Address and Transfer Data Primitive Format

Diese würde einen 68020/68030 anweisen, die in der FPU-Instruktion spezifizierten Daten, z. B. aus dem Speicher zu transferieren. Ohne hier auf alle Felder einzugehen, wird dennoch ersichtlich, dass die FPU einen Transfer zu ihr hin (DR = 0) mit einer Länge von 8 Bytes erwartet.

Es handelt sich hierbei um den Operanden *x* der zu berechnenden Arcuscosinus-Funktion acos(*x*), der als Double-Precision-Fließkommazahl 64 Bit (8 Byte) lang ist.

```
movel   %a7@(4),%a0@        OK >slpoke $FFFA50,0
movel   %a7@(8),%a0@

                            OK >slpoke $FFFA50,0
```

Das Argument wird folglich durch zwei Langwort-Transfers ins Operand-Register übertragen, zunächst die oberen 32 Bit, danach die unteren 32 Bit. Im dargestellten Beispiel wird die Zahl +0.0 übertragen, in deren Fließkomma-Darstellung alle Bits 0 sind.

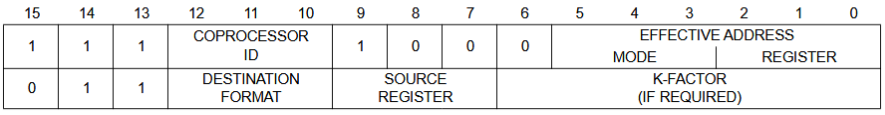
```
movew   #0x7400,%a0@(comm)          OK >sdpoke $FFFA4A,$7400
```

Der nächste Befehl wird in das Command-Register übertragen. Wieder liefert [4] dessen Bedeutung:

FMOVE **Move Floating-Point Data Register (MC6888X, MC68040)** **FMOVE**

Instruction Format:

REGISTER—TO-MEMORY



Es handelt sich demnach um einen FMOVE-Befehl aus einem FPU-Register zur CPU.[2] Das Datenformat ist erneut Double-Precision-Real (Destination Format = 101), das Quellregister ist FP0 (Source Register = 000), in dem das Ergebnis der vorangegangenen Berechnung abgelegt wurde.

```
.long   0x0c688900,0xfff067f8
... disassembliert also:
wait:
cmpiw   #0x8900,%a0@(resp)
beq.s   wait
```

```
OK >print hex$(dpeek($FFFA40))
8900

OK >print hex$(dpeek($FFFA40))
3208
```

Es wird wieder das Response-Register ausgelesen. Erneut erfolgt keine Fehlerbehandlung, es wird nur gewartet, bis der Wert ungleich $8900 ist.

Im Beispiel ist zu sehen, dass die FPU zunächst tatsächlich $8900 zurückliefert, eine *Null Primitive*:

7.4.2.1 NULL PRIMITIVE. This primitive is used by the FPCP to synchronize operation with the main processor and to allow concurrent execution by the main processor. The format of the null primitive is shown in Figure 7-8. In addition to the variable bits CA and PC previously discussed, the null primitive uses three other variable bits to identify the required action to be taken by the main processor. Bit [8], denoted by IA, is used to specify that the main processor may process

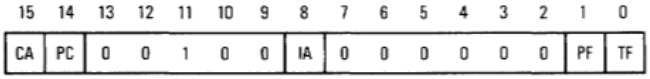


Figure 7-8. Null Primitive Format

Insbesondere zeigt CA = 1 (*come again*) an, dass das Register erneut gelesen werden soll. Der zweite Lesevorgang ergibt die bereits bekannte *Evaluate Effective Address and Transfer Data Primitive*. Jedoch wird dieses Mal mit $3208 ein Transfer von der FPU (DR = 1) von 8 Bytes Länge angezeigt. Dabei handelt es sich um die via FMOVE angefragten Daten des Registers FP0.

```
movel   %a0@,%d0
movel   %a0@,%d1
```

```
OK >print hex$(lpeek($FFFA50))
3FF921FB

OK >print hex$(lpeek($FFFA50))
54442D18
```

Die Daten werden – erneut als zwei Langworte – aus dem Operand-Register ausgelesen. Wie angefordert, handelt es sich um eine Double-Precision-Fließkommazahl, zunächst die oberen 32 Bit, danach die unteren 32 Bit.

Da die 68881/68882-FPU dem IEEE-754-Zahlenformat folgen, lässt sich die Zahl decodieren:

IEEE 754 Decoder

Hexadecimal	3FF921FB54442D18
Binary	0011111111111001001000011111101101010100010001000010110100011000
Exact Value	$+(1.1001001000011111101101010100010001000010110100011)_2 \times 2^0$
Printed Decimal Value (may be approximate)	1.5707963267948966

[2] Die Beschreibung des Befehls als *register to memory* ist aus Sicht der CPU zu verstehen. Die FPU hat keine Möglichkeit, Daten in den Speicher zu schreiben. Es ist Aufgabe der CPU, die *effective address* im ersten Wort des Opcodes zu decodieren und ggf. Daten in den Speicher zu schreiben.

Erkennbar handelt es sich um π/2, demnach um das korrekte Ergebnis der berechneten Funktion acos(0).

Mathematische Operationen mit zwei Operanden, z. B. eine Multiplikation, erfordern, dass ein Operand in einem FPU-Register vorhanden ist. Daher wird bei diesen Operationen ein Operand zunächst über einen *FMOVE <EA> to Register*-Befehl in die FPU geladen.

LITERATURVERZEICHNIS

[1] Motorola, Inc., MC68881/MC68882 Floating-Point Coprocessor User's Manual, Englewood Cliffs, NJ, USA: Prentice Hall, 1987.

[2] H.-D. Jankowski, D. Rabich und J. F. Reschke, „Anhang L: Der Coprozessor MC68881 im MEGA ST(E),“ in *Atari Profibuch ST-STE-TT*, Düsseldorf, Sybex-Verlag, 1992.

[3] Diverse Autoren, „Portable Math Library (PML),“ [Online]. Available: https://github.com/th-otto/pml/blob/01810cabc77c9540cc91d19b4c547232e8ba137e/pmlsrc/acos.c#L185-L196. [Zugriff am 2 August 2020].

[4] Motorola, Inc., M68000 Family Programmer’s Reference Manual, 1992.