Folie 1

Inhaltsverzeichnis 1

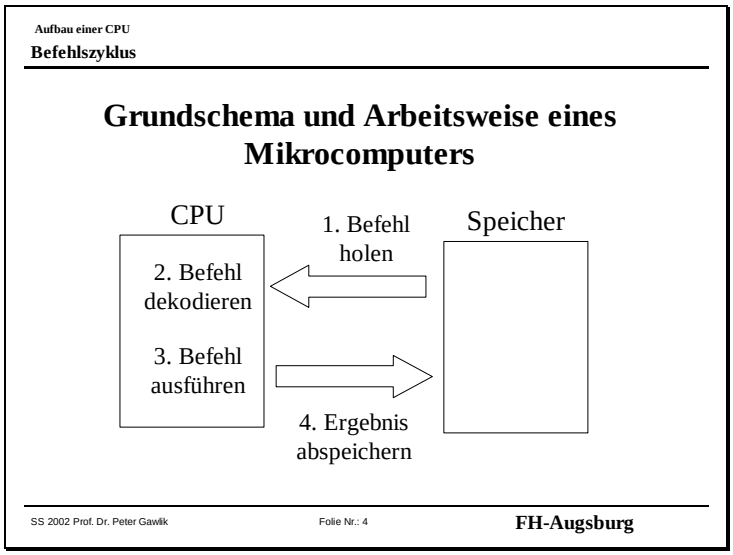
Folie 2

Inhaltsverzeichnis 2

Folie 3

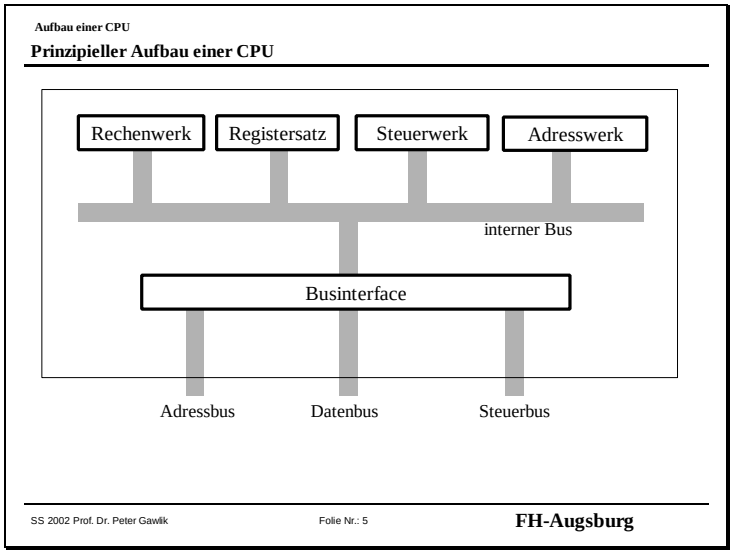
Inhaltsverzeichnis 2

Folie 4



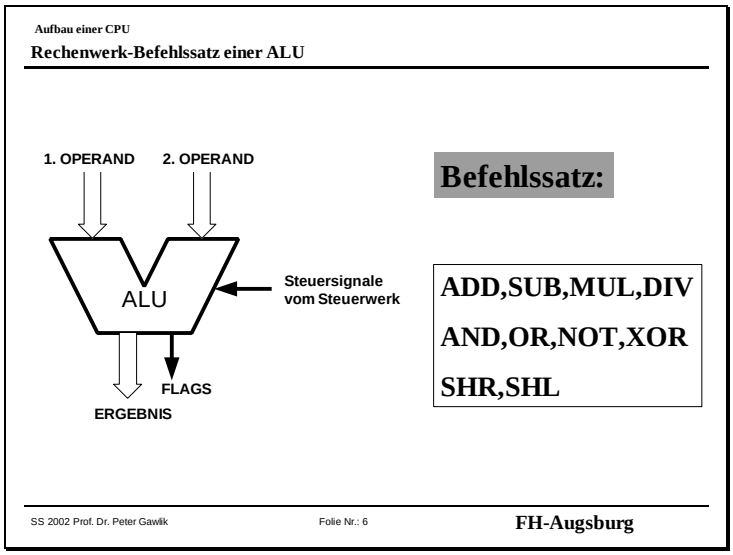
Jeder Befehl wird von der CPU in (mindestens) 4 Schritten bearbeitet.
Der Befehl wird in Form eines binären Datenwortes aus den Speicher eingelesen. Das binäre Wort wird als Befehl interpretiert (dekodiert). Der Befehl wird ausgeführt. Das Ergebnis der Befehlsausführung wird abgespeichert.

Folie 5



Zur Befehlsverarbeitung werden Daten im **Registersatz** zwischengespeichert. Die Informationsverarbeitung findet im **Rechenwerk** statt. Das **Steuerwerk** steuert alle internen und externen Abläufe, die zur Befehlsbearbeitung notwendig sind. Mit dem **Adresswerk** werden die Adressen für den Zugriff auf die Operanden berechnet. Heutige CPU`s erlauben es, auf die vielfältigste Art und Weise auf Operanden zuzugreifen. Die Methoden nennt man **Adressierungsarten**. Das **Businterface** sorgt für die Verbindung von internen und externen Bus. Eventuell müssen Datenbreiten und Spannungspegel angepasst werden

Folie 6



Die ALU wird als reines Schaltnetz ausgeführt, auch komplexe Rechenarten werden heute so ausgeführt. Neben den Dateneingängen und den Ergebnisausgängen (incl. Status oder Flags) erhält die ALU Steuer- und Selektsignale vom Steuerwerk. Zur Speicherung der Information werden je nach Realisierung verschiedene Register eingesetzt.
Der Befehlssatz der ALU hängt stark von der Realisierung ab. Er reicht von einigen wenigen Grundbefehlen bis zu sehr umfangreichen (>100) Befehlssätzen mit komplexester Arithmetik. Eine einfache 8-Bit-ALU kann heute durchaus im Rahmen eines Praktikums entwickelt werden.

ADD: Addiere 1. OP und 2. OP
SUB: Subtrahiere 1. OP und 2. OP
MUL: Multipliziere 1. OP und 2. OP
DIV: Dividiere 1. OP und 2. OP
AND: Verknüpfe 1. OP und 2. OP logisch UND
OR: Verknüpfe 1. OP und 2. OP logisch ODER
NOT: Bilde die logische Invertierung von 1. OP
XOR: Verknüpfe 1. OP und 2. OP logisch XOR
SHR: Schiebe die Bits des 1. OP um so viele Bits nach rechts wie 2. OP angibt
SHL: Schiebe die Bits des 1. OP um so viele Bits nach links wie 2. OP angibt

Folie 7

Aufbau einer CPU

Rechenwerk-Befehlssatz einer ALU

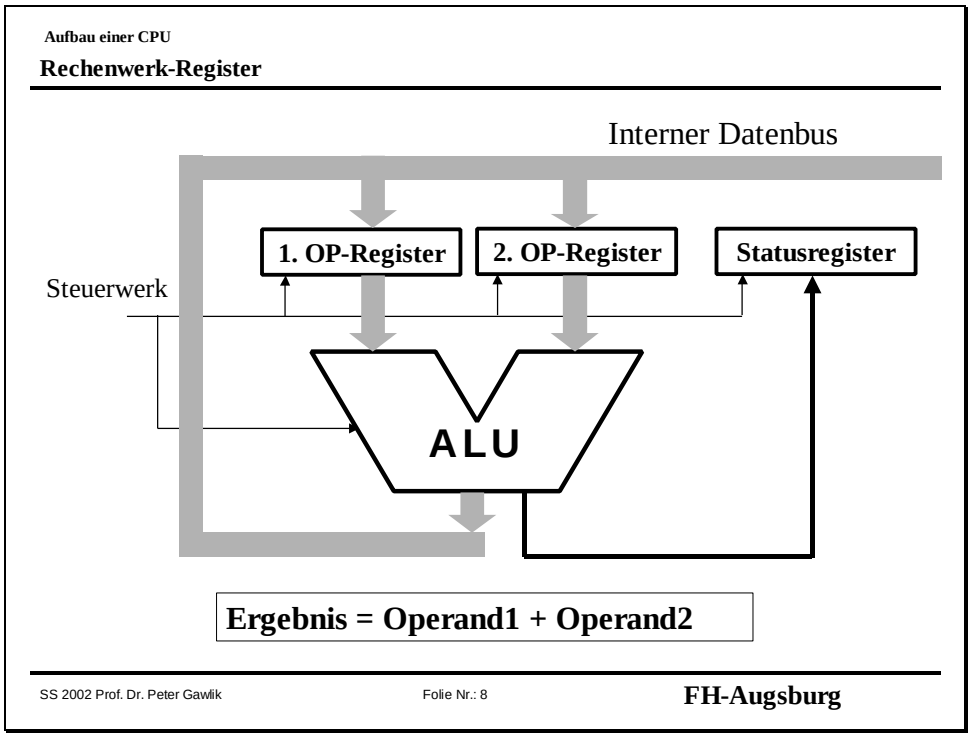
				M = H	M = L; Arithmetic Operations	
				Logic	/Cn = H	/Cn = L
S3	S2	S1	S0	Functions	(No Carry)	(With Carry)
L	L	L	L	F = /A	F = A	F = A plus 1
L	L	L	H	F = /(A + B)	F = A + B	F = (A + B) plus 1
L	L	H	L	F = (/A)B	F = A + /B	F = (A + /B) plus 1
L	L	H	H	F = 0	F = minus 1 (2s Comp)	F = ZERO
L	H	L	L	F = /(AB)	F = A plus A(/B)	F = A plus A(/B) plus 1
L	H	L	H	F = /B	F = (A + B) plus A(/B)	F = (A + B) plus A(/B) plus 1
L	H	H	L	F = A $ B	F = A minus B minus 1	F = A minus B
L	H	H	H	F = A(/B)	F = A(/B) minus 1	F = A(/B)
H	L	L	L	F = /A+ B	F = A plus AB	F = A plus AB plus 1
H	L	L	H	F = /(A $ B)	F = A plus B	F = A plus B plus 1
H	L	H	L	F = B	F = (A + /B) plus AB	F = (A + /B) plus AB plus 1
H	L	H	H	F = AB	F = AB minus 1	F = AB
H	H	L	L	F = 1	F = A plus A*	F = A plus A plus 1
H	H	L	H	F = A + /B	F = (A + B) plus A	F = (A + B) plus A plus 1
H	H	H	L	F = A + B	F = (A + /B) plus A	F = (A + /B) plus A plus 1
H	H	H	H	F = A	F = A minus 1	F = A

Befehlssatz einer 4-Bit-ALU (74181)

SS 2002 Prof. Dr. Peter Gawlik Folie Nr.: 7 **FH-Augsburg**

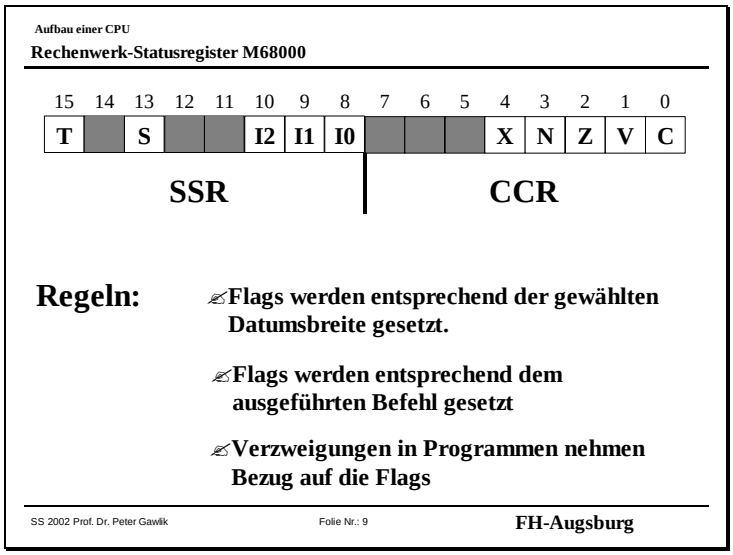
Die 4-Bit-ALU kann zu einer 8-Bit-ALU kaskadiert werden. Mit ca. 150 Gattern lässt sich diese ALU realisieren. Von der Komplexität entspricht sie in etwa einer ALU in einer heute gängigen 8-Bit-CPU. Wenn keine hohen zeitlichen oder Flächenverbrauchsanforderungen gestellt werden, kann man eine ALU in einer Hochsprache beschreiben (wurde in unserem Praktikum Digitaltechnik 2 so gemacht). Werden allerdings hohe Anforderungen gestellt, muss man sehr genau darauf achten, was der Compiler (Übersetzer) aufgrund welcher sprachlichen Konstrukte an Hardware erzeugt.

Folie 8



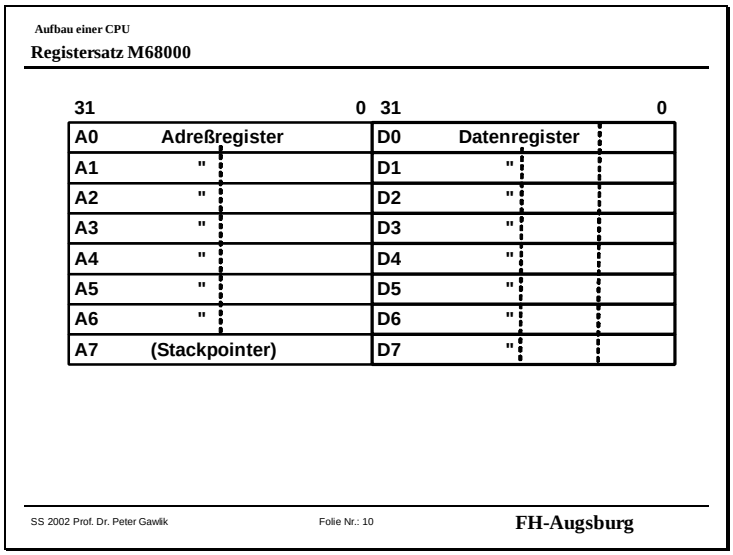
Gesteuert durch den Befehlscode, werden der Erste- und der Zweite Operand in das jeweilige Register transportiert. Das Ergebnis kann z.B. in das 1. Operandenregister geschrieben werden. Viele CPU haben zur Zwischenspeicherung von Daten Registersätze. Acht Register ist eine gängige Anzahl. In diesem Fall dienen die Rechenwerksregister nur als Hilfsregister und sind nicht direkt vom Programmierer ansprechbar.
Die ALU erzeugt neben dem Ergebnis noch weitere Informationen, die für die Fortführung eines Programms wichtig sein können. Diese Informationen nennt man Statusinformation oder Flags. Einige wichtige Flags sind:
C: C- Flag (Carry-Flag) gibt an, ob bei einer arithmetischen Operation ein Überlauf aufgetreten ist.
V: V-Flag (Overflow-Flag) zeigt an, ob bei einer arithmetischen Operation im 2-er-Komplement ein Überlauf aufgetreten ist.
Z: Z-Flag (Zero-Flag) zeigt an, wenn das Ergebnis einer Operation gleich Null war.
N: N-Flag (Negativ-Flag) zeigt an, wenn das Ergebnis einer Operation negativ war (N ist eine Kopie des MSB. In den meisten Zahlendarstellungen vorzeichenbehafteter Zahlen, wird mit MSB=1 ein negativer Wert angezeigt).
Der hier gezeigte Aufbau ist nur beispielhaft zu sehen. Immer müssen die zu verarbeitenden Daten in der CPU zwischengespeichert werden. Von einfachen bis zu komplexen CPU‘s gibt es viele Varianten. Bei sehr einfachen CPU‘s gibt es nur einen Zwischenspeicher. In diesem muss der 1. Operand und das Ergebnis gespeichert werden. Der zweite Operand liegt permanent auf den Internen Datenbus. Solche CPU‘s nennt man Akkumulator-Maschinen. Der Name stammt aus der Eigenschaft, dass bei verketteten arithmetischen Operationen im Akkumulator das Ergebnis akkumuliert wird.

Folie 9



Das Statusregister des M68000 ist 16-Bit breit. Nicht alle Bits werden verwendet. Die nicht verwendeten Bits sind grau hinterlegt. In der Regel setzt der Hersteller diese Bits auf den Wert „0“. Das **Statusregister (SR)** ist zweigeteilt. Das **SSR (System-Status-Register)** wird im sogenannten Supervisor-Mode verwendet. In diesem Modus sind einige privilegierte Befehle zugänglich. Hierdurch wird es möglich, bestimmte Systemteile (z.B. Betriebssystem) zu schützen. Die Bits im **CCR** sind:
T:Trace-Mode (=1) zeigt an, dass µP nach jedem Befehl zu einer bestimmten Adresse springen soll. Hierdurch kann man im Testbetrieb einen sogenannten Single-Step-Betrieb durchführen.
S: Supervisor-Mode (=1) zeigt an, dass der µP sich im Supervisor Mode befindet. Hier stehen priviligierte Befehle zur Verfügung. Nach dem Einschalten befindet sich der µP im Supervisor Modus.
I2..I0 Interruptmaske Auf die Verwendung dieser Bits gehen wir später ein.
Die Bits im **CCR (Condition-Code-Register)** wurden bis auf das X-Bit bereits besprochen.
X: EXtend erweitert. Bei arithmetischen Operationen wird **X** wie **C** gesetzt. Bei den restlichen Operationen wird das **X**-Bit nicht beeinflusst. Es dient z. B. als Merker für das **C**-Bit bei Kettenadditionen.

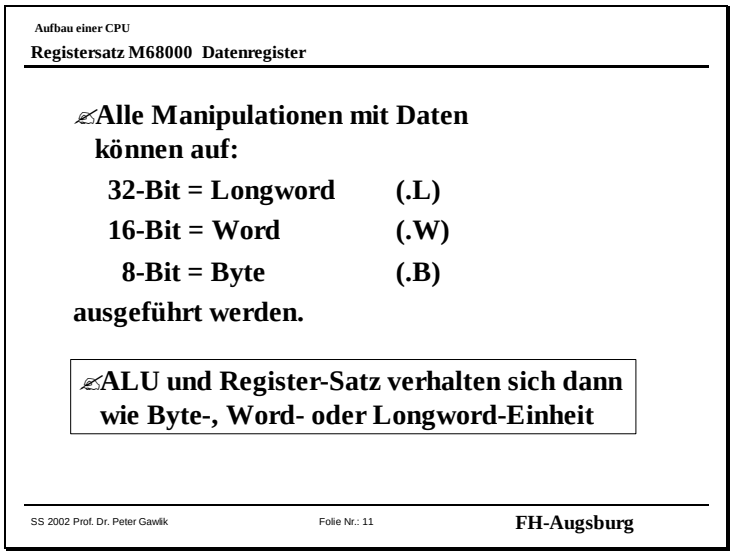
Folie 10



Der Registersatz dient zur Informationszwischenspeicherung.
Die **Datenregister** sind im Prinzip wie die Speicher bei einem Taschenrechner zu verwenden. Die Datenbreite kann dabei 8-Bit, 16-Bit und 32-Bit betragen. Die Befehle des M68000 erlauben die Angabe der gewünschten Datenbreite. Die ALU berücksichtigt die verwendete Datenbreite und passt alle Operationen entsprechend an. So

werden bei einer 8-Bit Operation die Statusbits entsprechend dieser Breite gesetzt. Die anderen Bits des jeweiligen Datenregisters bleiben unbeeinflusst.
In den **Adressregistern** werden hauptsächlich Adressen von Operanden (Zeiger) zwischengespeichert. Mit ihnen lässt sich dann effektiv auf den Speicher zugreifen. Die möglichen Breiten sind 16 und 32 Bit.
Das Register A7 hat eine Spezialaufgabe. Es dient als sogenannter „Stackpointer“ (Stapelzeiger).
Die Adressregister können auch zur „normalen Datenspeicherung“ verwendet werden. Es sind ist jedoch die eingeschränkten arithmetischen Möglichkeiten zu beachten. Auch die Datenbreite können Probleme machen.

Folie 11

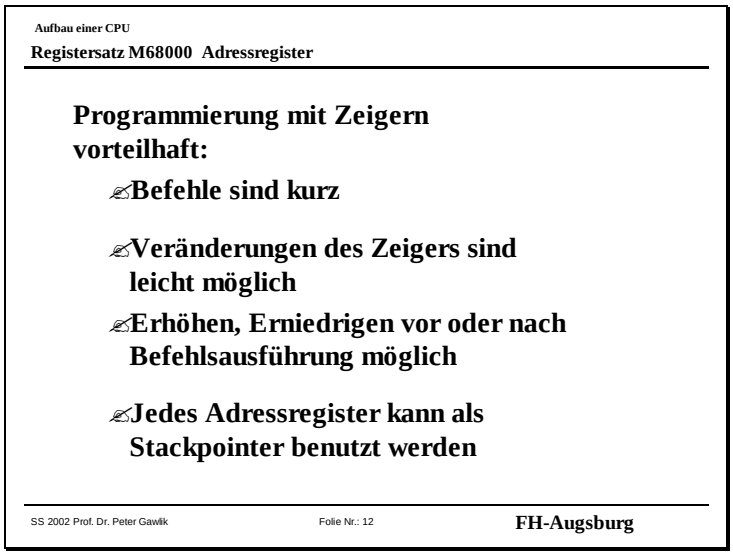


Die Bezeichnungen (.L), (.W) und (.B) beziehen sich auf eine Schreibweise in der Assemblerprogrammierung. Diese Erweiterungen (extensions) werden den entsprechenden Befehlsabkürzungen angehängt. Beispiel:

ADD.B #$73,D0

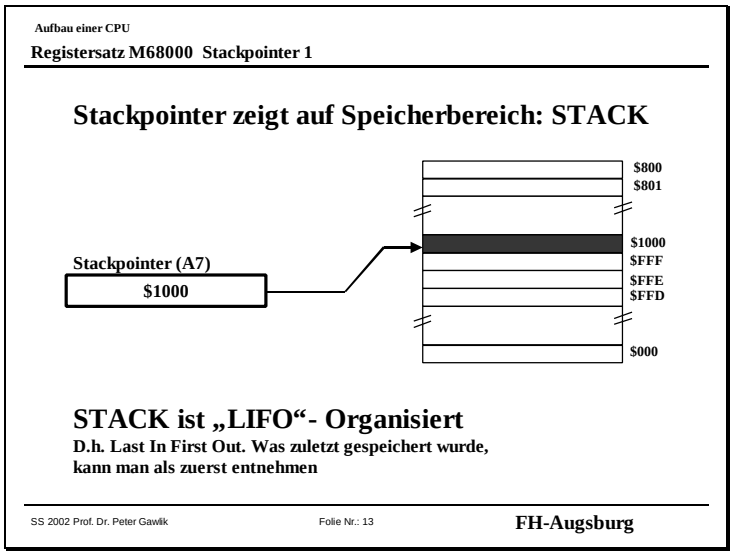
Addiere im Datenformat Byte zum Inhalt des Datenregisters 0 die Zahl $73.
Beträgt der 32-Bit Inhalt von D0 z.B. D0 = $00 34 18 F3, so ist das Ergebnis
D0 = $00 34 18 66
Es wird also nur das unterste (rechte) Byte beeinflusst. In diesem Fall tritt ein Überlauf auf. Entsprechend wird das C-Bit gesetzt.

Folie 12



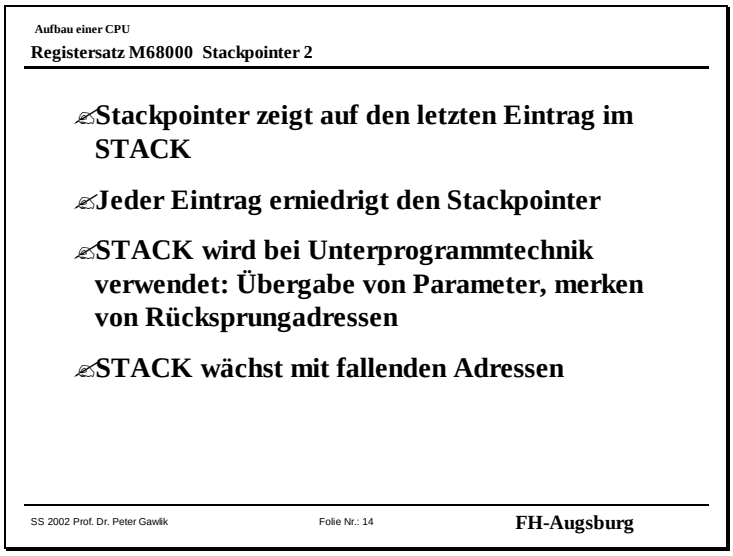
Die Möglichkeiten des Datenzugriffs werden wir noch bei der Diskussion des Adresswerks und bei der Programmierung näher betrachten.

Folie 13



Die Folie zeigt eine Situation bei der im Stackpointer $1000 steht. Soll nun ein weiteres Datum auf den STACK geschrieben werden, wird zunächst der Stackpointer um so viele Bytes erniedrigt, wie das zu speichernde Datum enthält und dann wird das Datum auf den STACK geschrieben. Das Holen eines Datums vom STACK erfolgt ebenso automatisch. Der Stackpointer wird dabei entsprechend erhöht.
Wir werden den STACK im Abschnitt Programmieren noch genauer kennen lernen.
Anmerkung: Beim M68000 können alle Adressregister als Stackpointer verwendet werden. Das Register A7 zeichnet sich dadurch aus, dass es automatisch verwendet wird und einige Befehle sich direkt auf A7 beziehen.

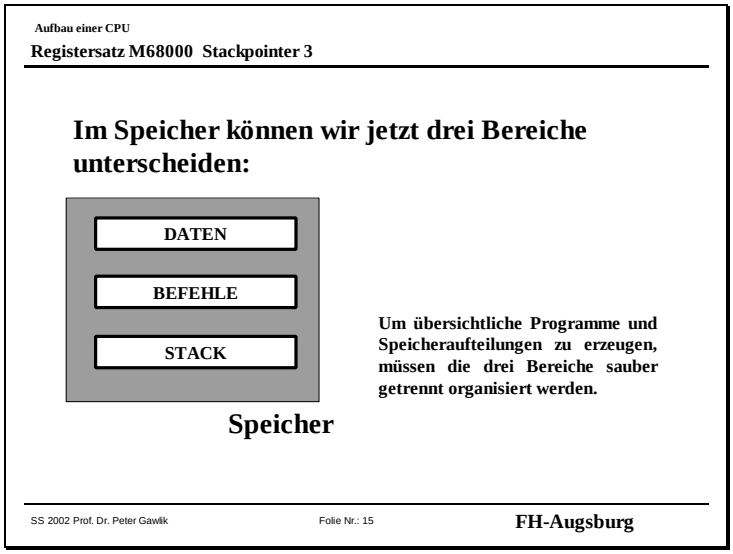
Folie 14



Ohne Veränderung des Stackpointers wird direkt auf den letzten Eintrag zugegriffen z. B. move.l (A7),D0.
Der schreibende Zugriff kann durch unterschiedliche Befehle erfolgen (ebenso das Lesen).

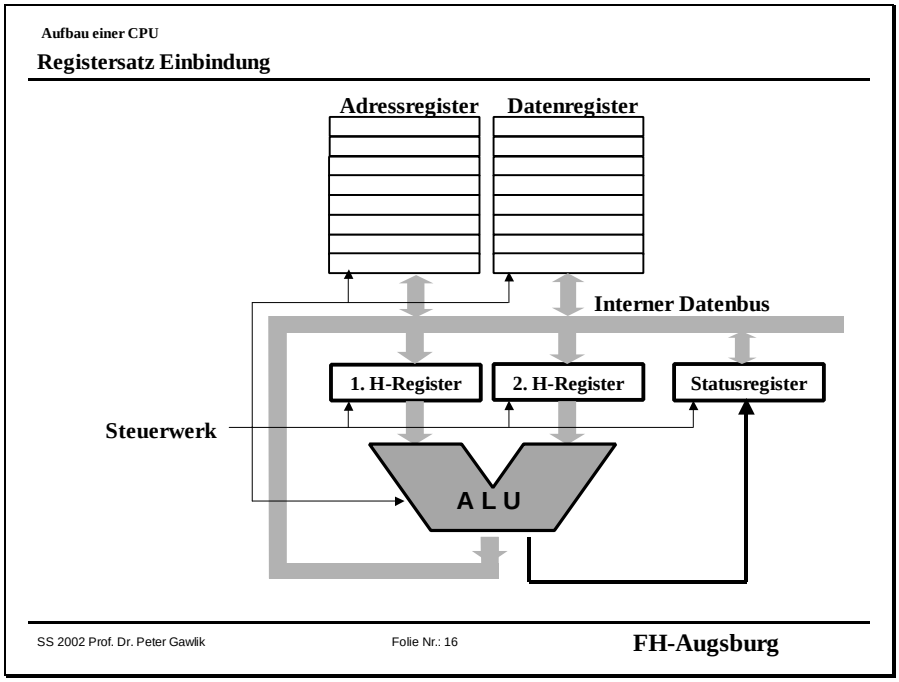
Bei vielen Systemen wächst der STACK mit fallenden Adressen. Es ist aber genauso möglich, einen STACK mit steigenden Adressen zu definieren.
Der STACK-Bereich ist für ein System festgelegt. Wird der Bereich verlassen so werden Daten oder Programme zerstört, was meist zum Systemabsturz führt. Bei modernen Systemen wird der Stackbereich durch die MMU überwacht.

Folie 15



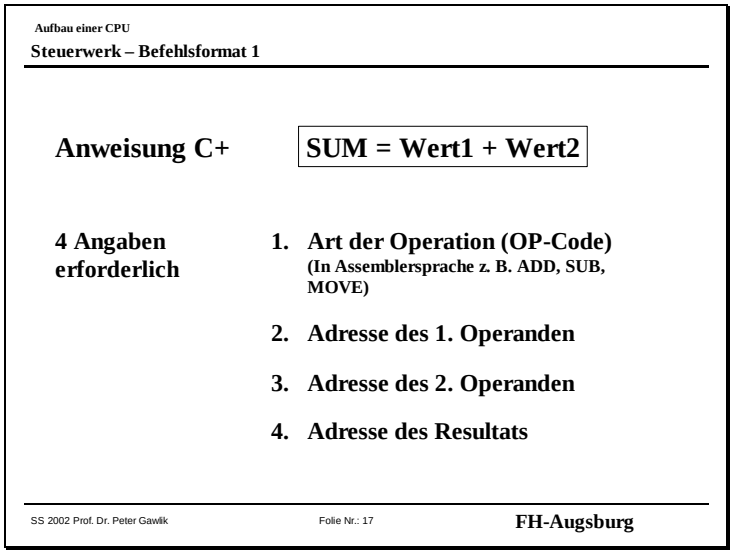
In Entwicklungssystemen – Das sind Programmpakete zum Erstellen und Austesten von Assembler- oder Hochsprachen-Programmen –
Werden die drei Bereiche dem Entwickler bereits getrennt angeboten. Der Entwickler muss nur noch die vorhandenen Speicheradressen und die Größe der jeweiligen Blöcke eintragen.

Folie 16



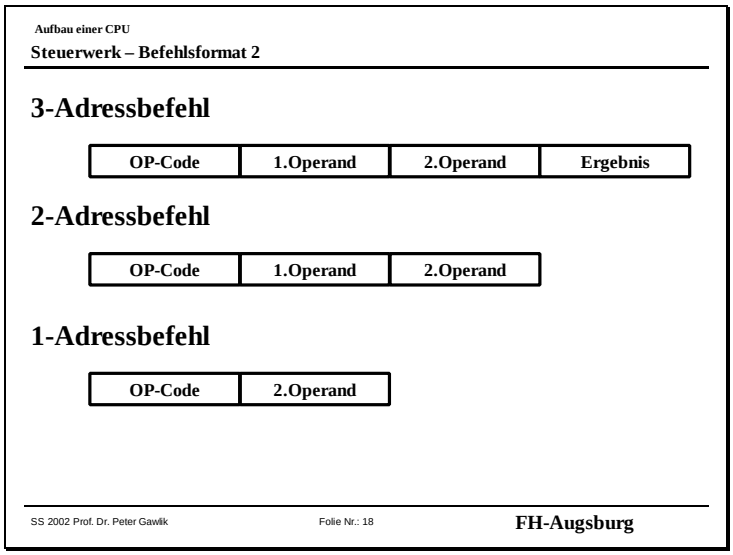
Die Einbindung des Registersatzes erfolgt über den Internen Datenbus. Die Hilfsregister 1. H und 2.H sind vom Programmierer nicht ansprechbar.

Folie 17



Was das Steuerwerk tun muss, hängt stark vom inneren Aufbau und dem Befehlsformat ab. So ist es unmittelbar einsichtig, dass bei einer CPU mit nur einem Zwischenspeicher (AKKU) für Daten andere innere Abläufe bei der Befehlsabarbeitung notwendig sind als z. B. bei dem hier gezeigten Aufbau. Die Struktur der Befehle (Befehlsformat) hängt eng mit dem inneren Aufbau einer CPU zusammen.
Befehlsformate mit denen man direkt Hochsprachenbefehle umsetzen kann, sind bei modernen Prozessoren üblich. Soll ein solcher Hochsprachenbefehl hingegen mit einer CPU mit nur einem Akkumulator realisiert werden, ist eine Zerlegung des Hochsprachenbefehls in für diese CPU besser angepasste Befehle notwendig.

Folie 18



3-Adressbefehl: Eine solche Befehlsstruktur ist leicht zu verstehen aber die Befehle werden sehr lang (10 Byte und mehr). Weiter sind viele Speicher-zugriffe erforderlich, was die Ver-

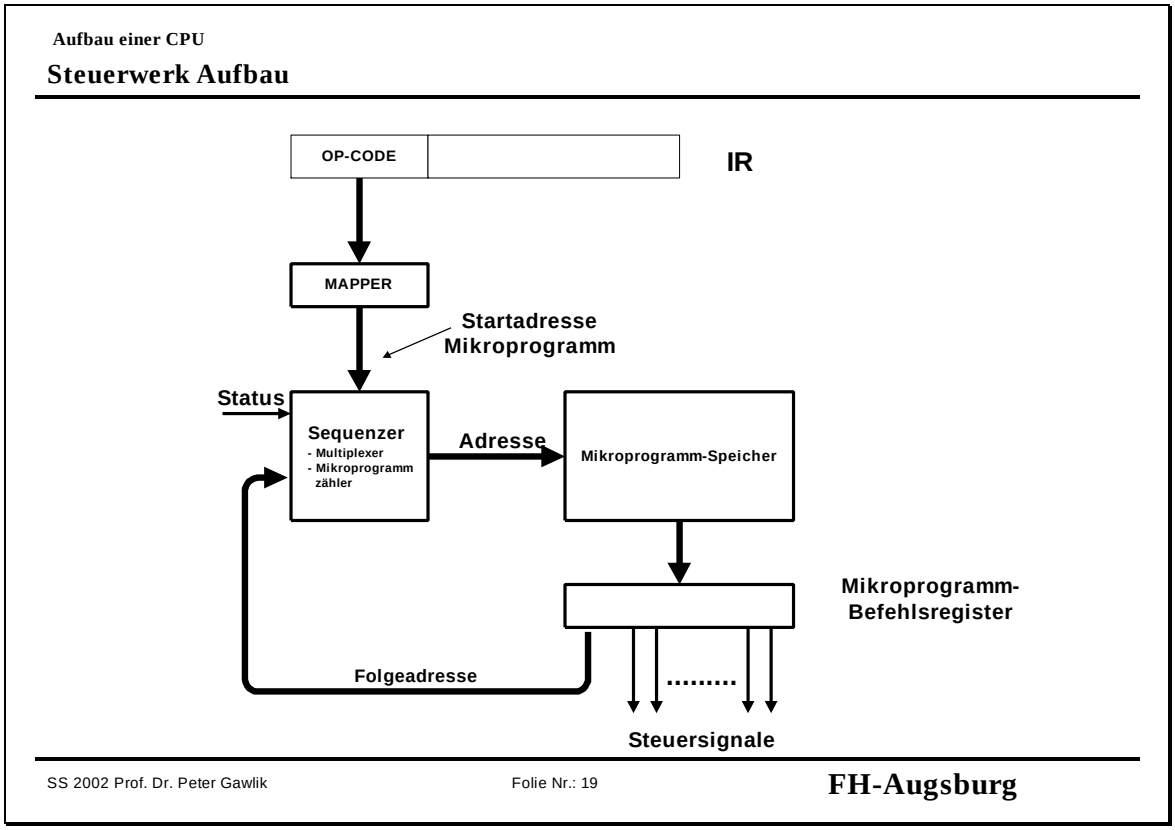
arbeitungsgeschwindigkeit herabsetzt. In vielen Fällen sind nicht alle Angaben erforderlich. So kann man den Befehls-umfang reduzieren, wenn als Zieladresse für das Ergebnis die Adresse des 1. Operanden benutzt wird. Dies führt auf den

2-Adressbefehl: Der Befehlsumfang kann noch weiter reduziert werden, wenn festgelegt wird, dass der 2. Operand in einem bestimmten Register zur Verfügung steht, und das Ergebnis in dasselbe Register gespeichert wird. Dies geht natürlich auf Kosten der Verarbeitungsgeschwindigkeit.

1. Adressbefehl: Strukturen mit AKKU (Akkumulator)

1. OP ✍ AKKU ; Ergebnis ✍ AKKU

Folie 19



Bei den meisten Prozessoren wird heute das Steuerwerk als sogenanntes Mikroprogramm-Steuerwerk (µPStw) realisiert. Das µPStw stellt einen speziellen Prozessor dar, dessen Aufgabe es ist, Steuersequenzen zu erzeugen.

Der OP-Code des Befehlswortes wird mit Hilfe des **Mappers** in eine Start-adresse für eine Mikrobefehlssequenz umkodiert. Der Sequenzer erzeugt Adressen für den Mikroprogrammspeicher. Die Mikrobefehlssequenzen stehen im **Mikroprogrammspeicher**. Jeder Mikrobefehl besteht aus einzelnen Bits, die die Steuersignale für den Prozessor darstellen. In der Regel wird hier nicht mehr kodiert sondern jedes Bit stellt direkt ein Steuersignal z.B. ein Freigabesignal für ein Register oder ein Auswahlsignal dar.

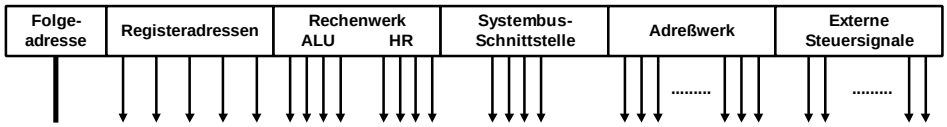
Neben dem µPStw gibt es auch festverdrahtete Steuerwerke. Diese setzen aber einfach Prozessorstrukturen und Befehlsstrukturen voraus. Diese Anforderung wird von den RISC-Prozessoren erfüllt. RISC bedeutet i **R**educed **I**nstruction **S**et **C**omputer.

Der hier besprochene Prozessor ist eher der alternativen Klasse der CISC-Prozessoren zuzuordnen. CISC bedeutet Complex Instruction Set Computer.

Folie 20

Steuerwerk Mikrobefehl

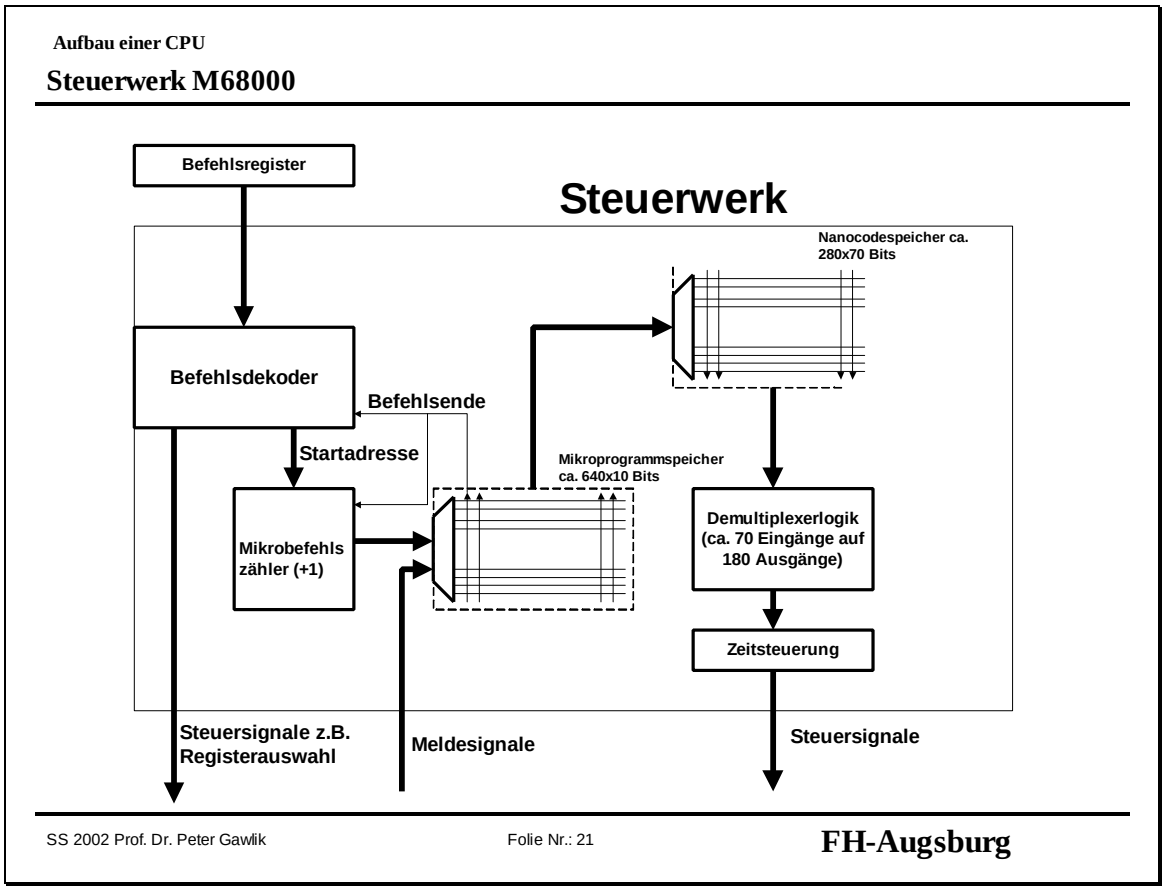
Aufbau eines Mikrobefehls



- **Steuerwerk ist synchrones Schaltwerk**
- **1/3 – 2/3 der Chipfläche werden für Steuerwerk benötigt!**

Dementsprechend besteht ein Mikrobefehl aus 70..150 Bits.
Die Folgeadresse für einen Mikrobefehl wird aus einem Teil des Mikrobefehls und den Statusinformationen gewonnen, so dass das Mikroprogramm nicht nur lineare Sequenzen durchlaufen kann, sondern auch Sprünge realisierbar sind.

Folie 21



Über den internen Datenbus gelangt der Befehl in das Befehlsregister. Der Befehlsdecoder wertet den Befehl aus und erzeugt eine 10 Bit Startadresse für ein Mikroprogramm. Der Mikrobefehlszähler wird auf diese Adresse geladen. Signale, die nicht vom Mikrobefehl aktiviert werden müssen z. B. Register-Selekt- Leitungen und ALU-Kommandos werden direkt weitergeleitet.
Das µPStw ist zweistufig organisiert. Es zeigt sich nämlich, dass bei vielen Mikroprogrammen Befehlssequenzen identisch sind. Durch die zweistufige Realisierung brauchen diese nur einmal abgelegt werden. In den 640 10-Bit-Worten sind sämtliche Befehle des Prozessors kodiert. Die eigentlichen Steuersignale werden im Nanospeicher erzeugt, der etwa 27 KBit groß ist. Die Zeitsteuerung sorgt dafür, dass die Signale zu den richtigen Zeitpunkten weitergeleitet werden.

Folie 22

Aufbau einer CPU

Adresswerk Aufgabe

Berechnet die sogenannte „Effektive Adresse“

Beispiel: **MOVE D2, $47(A0,D1.B)**

Basisadresse im Register A0

+ 8-Bit-Abstand in D1

+ Offset $47 aus Befehl

= Effektive Adresse

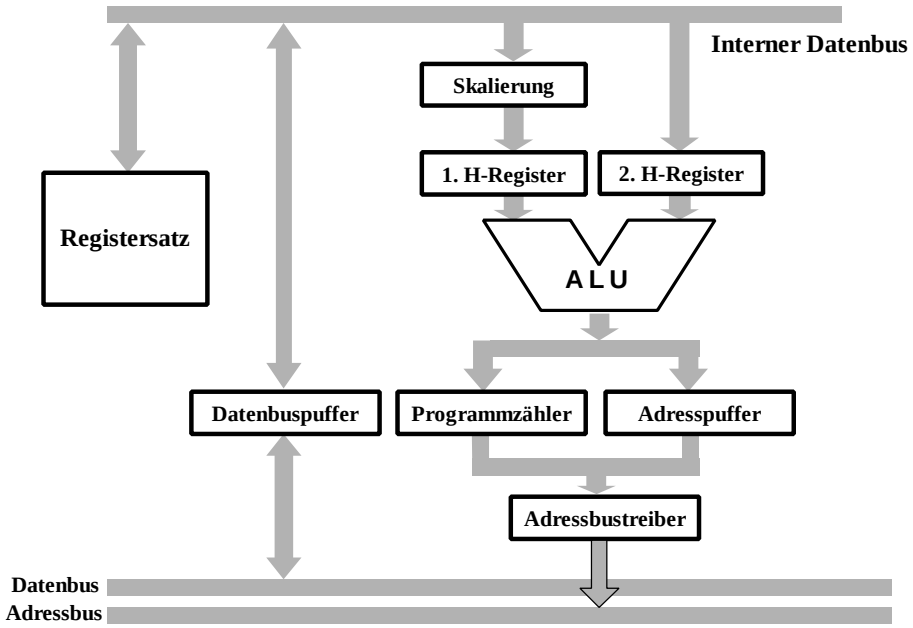
Effektive Adresse: Physikalisch am Speicher anliegende Adresse

Wie alle anderen CPU-Teile auch, variieren Adresswerke von einfach bis komplex. Die Komplexität ist so groß geworden, dass für die Aufgabe der Adressberechnung ein separater Chip, die Memory Management Unit (MMU), entwickelt wurde. Die MMU entlastet die CPU und unterstützt Betriebs-systeme bei der Speicherverwaltung. Heute wird die MMU meist auf dem CPU-Chip mit integriert.

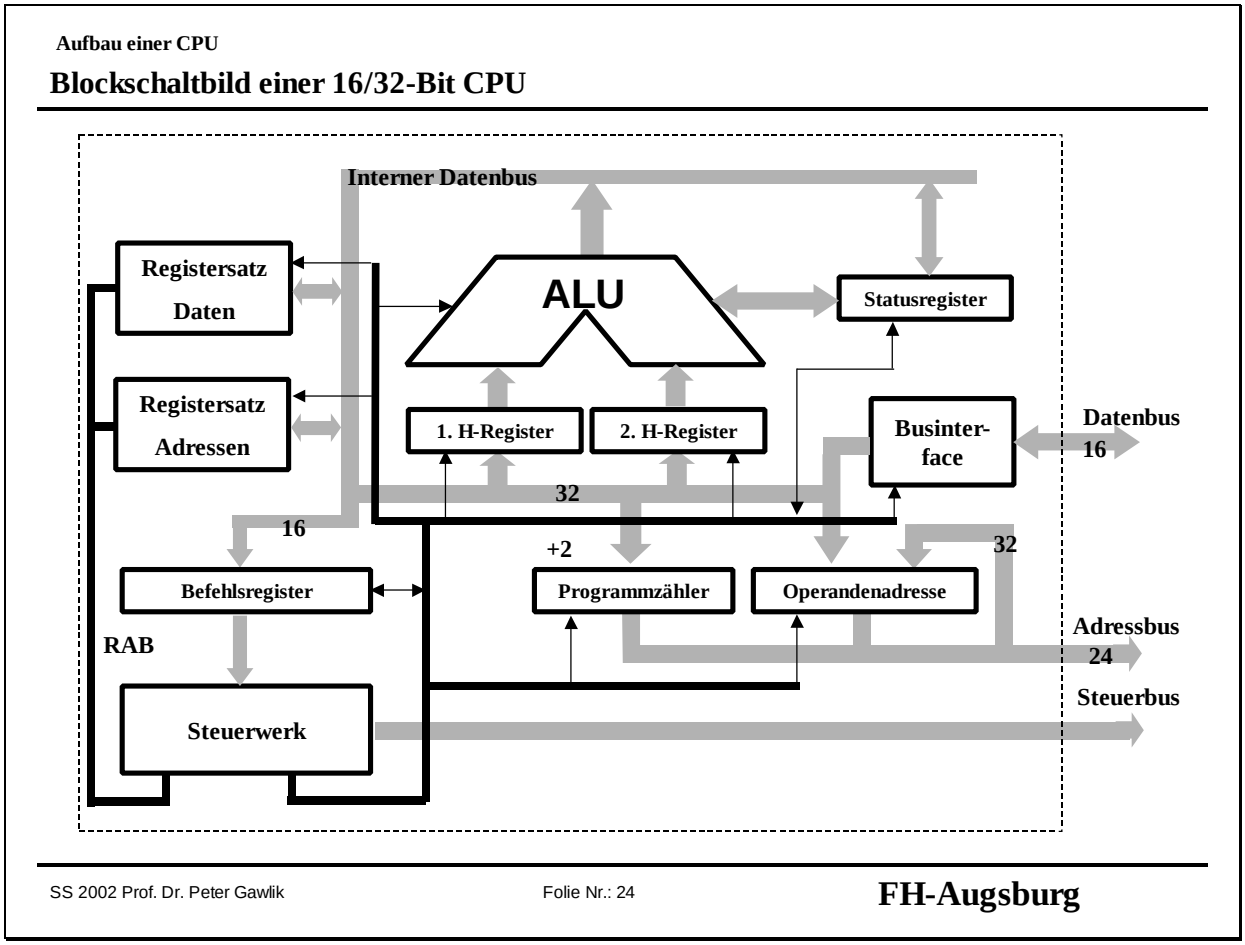
Folie 23

Aufbau einer CPU

Adresswerk Einfaches

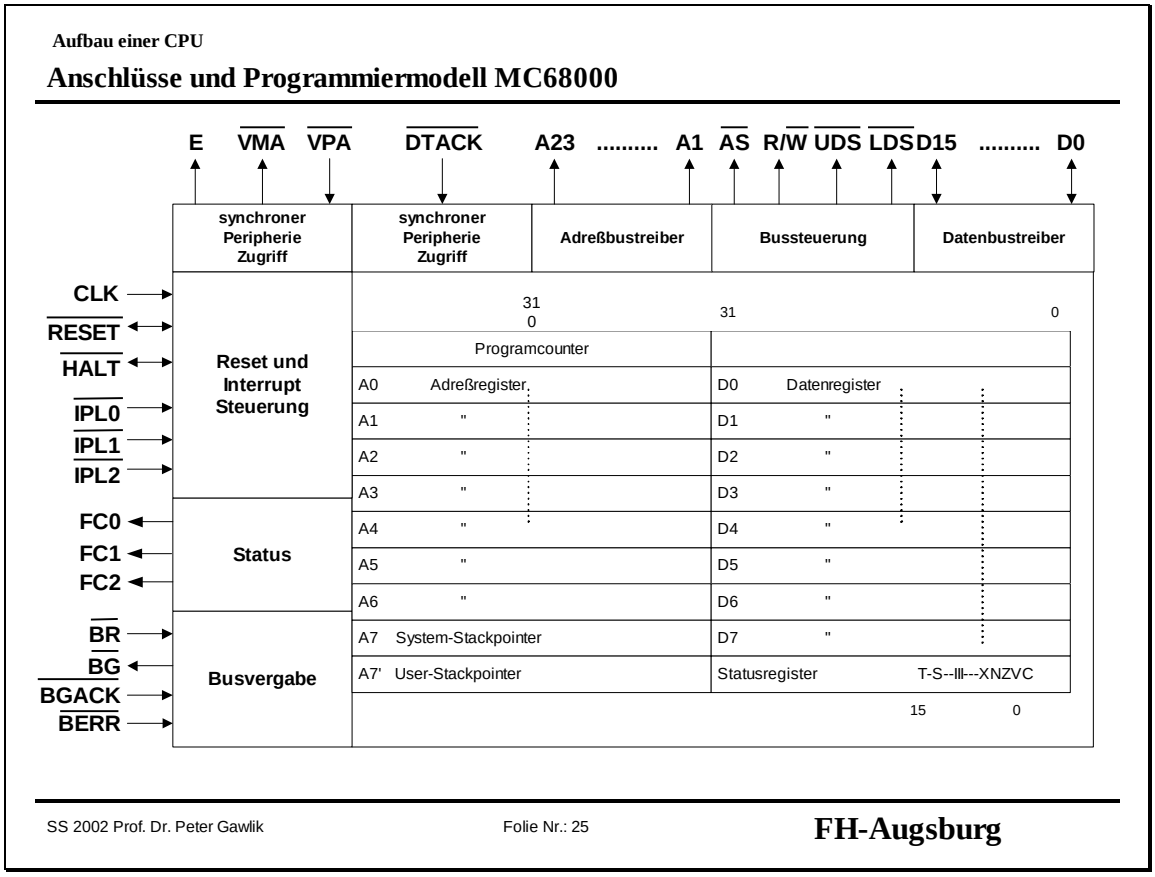


Die Folie zeigt ein einfaches Adresswerk, wie es z.B. im µP M68000 oder 8086 eingebaut sein könnte. Die Strukturen unterscheiden sich gar nicht so stark vom Rechenwerk. Wenn die Subtraktion im 2-er-Komplement ausgeführt ist, muss die ALU nur addieren. Die Skalierung erzeugt Adressverschieb-ungen, die beim Speicherzugriff mit unterschiedlichen Datentypen (Byte, Word, Longword) notwendig sind. Wie das vorherige Beispiel zeigt, sind die Quellen für die Adressberechnung der Registersatz und Befehle, die im Speicher stehen ($47 beim Befehl MOVE D2, $47(A0,D1.B) steht im Speicher).
Auf den Adressbus kann der Programmzähler oder der Adresspuffer geschaltet werden.
Der Befehlszähler zeigt immer auf die Adresse des nächsten zu holenden Befehls und wird bei einer zyklischen Befehlsbearbeitung automatisch erhöht (inkrementiert). Dieser Vorgang wird nur bei programmierten Sprüngen oder Unterprogrammsprüngen unterbrochen. In diesen Fällen muss der Programm-zähler auf einen neuen Wert geladen werden.
Der Adresspuffer wird zur Adressierung der Daten (Operanden) verwendet. In ihm steht die Effektive Adresse.
Folie 24



Das Blockschaltbild fasst die bisherigen Ausführungen zusammen. Der Aufbau ist näherungsweise mit dem µP MC68000 zu vergleichen. Damit die Übersicht gewahrt bleibt, wurden einige CPU-Teile stark vereinfacht dargestellt. So wurde das Adresswerk auf Programmzähler und Operandenadresse reduziert.
Bei dieser CPU handelt es sich um einen 16/32-Bit Prozessor. Der externe Datenbus ist 16-Bit breit und der externe Adressbus ist 24-Bit breit. Intern ist die CPU auf 32-Bit ausgelegt.
Ein Befehl (immer nx16-Bit breit) gelangt über das Businterface in das Befehlregister. Das Steuerwerk dekodiert den Befehl und steuert alle weiteren für die Befehlsausführung notwen-

digen Aktionen aus. Werden im Befehl Register angesprochen, werden diese über den Register-Adress- Bus (RAB) abgewählt. Operanden im Speicher werden über das Businterface in die entsprechenden Register gebracht. Verschiedene Datenbreiten werden vom Businterface und der ALU entsprechend behandelt.
Der Programmzähler adressiert den nächsten Befehl (oder Teilbefehl). Alle Befehle haben eine Größe von Vielfachen von 16. Aus diesem Grund zählt der Programmzähler immer um 2 hoch (jede Adresse belegt ein Byte), wenn er nicht mit einem neuen Wert geladen wird (Sprung).
Folie 25



Zum Schluss ein Überblick über alle Anschlüsse und Register des MC68000.
Eine Darstellung aller Register einer CPU wird mit Programmiermodell bezeichnet. Das Programmiermodell reicht in der Regel zum Programmieren einer CPU aus.
Neben den Signalen sind auch die zuständigen Funktionsgruppen dargestellt.

Datenbustreiber:	Stellen ausreichend Leistung zum Ansteuern mehrerer Einheiten zur Verfügung.
Bussteuerung:	Teil des Steuerwerks, übernimmt die Abwicklung des Datentransfers.
Adressbustreiber:	Stellen ausreichend Leistung zum Ansteuern mehrerer Einheiten zur Verfügung.
Synchroner Zugriff, Peripherie:	Teil des Steuerwerks, Anschluss von alten 8-Bit Modulen, Steuerung von Wartezyklen
Reset und Interruptsteuerung:	Unterbrechung und externe Steuerung der Befehlsbearbeitung
Status:	Informationen über den CPU Status. Bedeutung der Adressleitungen

Busvergabe: Übergabe der Systembuskontrolle an andere Einheiten (CPUs, DMA).

Folie 26

Aufbau einer CPU

Signalnamen MC68000

Table 3-4. Signal Summary

Signal Name	Mnemonic	Input/Output	Active State	Hi-Z	
				On HALT	On Bus Relinquish
Address Bus	A0–A23	Output	High	Yes	Yes
Data Bus	D0–D15	Input/Output	High	Yes	Yes
Address Strobe	AS	Output	Low	No	Yes
Read/Write	R/W	Output	Read-High Write-Low	No	Yes
Data Strobe	DS	Output	Low	No	Yes
Upper and Lower Data Strobes	UDS, LDS	Output	Low	No	Yes
Data Transfer Acknowledge	DTACK	Input	Low	No	No
Bus Request	BR	Input	Low	No	No
Bus Grant	BG	Output	Low	No	No
Bus Grant Acknowledge	BGACK	Input	Low	No	No
Interrupt Priority Level	IPL0, IPL1, IPL2	Input	Low	No	No
Bus Error	BERR	Input	Low	No	No
Mode	MODE	Input	High	—	—
Reset	RESET	Input/Output	Low	No*	No*
Halt	HALT	Input/Output	Low	No*	No*
Enable	E	Output	High	No	No
Valid Memory Address	VMA	Output	Low	No	Yes
Valid Peripheral Address	VPA	Input	Low	No	No
Function Code Output	FC0, FC1, FC2	Output	High	No	Yes
Clock	CLK	Input	High	No	No
Power Input	VCC	Input	—	—	—
Ground	GND	Input	—	—	—

Folie 27

Zusammenfassung

CPU besteht aus:	**Rechenwerk, Steuerwerk, Adresswerk, Interface**
Rechenwerk besteht aus:	**ALU, Registersatz, Statusregister**
ALU:	**Reines Schaltnetz, heute komplex**
Registersatz:	**AKKU ✍ viele Universalregister**
Statusregister:	**Bedingungsbits, Sonderflags**
Adresswerk:	**indizierte Adressierung ✍ MMU**
Steuerwerk:	**synchrones Schaltwerk, als µPStw realisiert**
Interface:	**passt Datenbreiten und Pegel an**

Folie 28

Übungen

1. Wie ist ein mikroprogrammiertes Steuerwerk aufgebaut?
2. Was ist ein Sequenzer?
3. Was steht in einem Mikroprogrammspeicher?
4. Mit welchen Aktionen kontrolliert das Steuerwerk die Informationsverarbeitung?
5. Welche Aufgabe hat das Mikroprogrammbefehlsregister
6. Warum wird beim MC68000 Statusregister zwischen User-Byte und Supervisor-Byte unterschieden?
7. Wozu dient das V Bit im Statuswort?
8. In welche Kategorien lassen sich ALU-Operationen einteilen?
9. Wozu dient der Stackpointer?
10. Für welches Befehlsformat ist die hier vorgestellte CPU besonders geeignet?
11. Wozu dienen die Datenregister im Rechenwerk?
12. Wozu dienen die Adressregister im Rechenwerk?

Übungen

Aufgabe 1: Bauen Sie mit den folgenden Komponenten ein Mikroprogramm - Steuerwerk auf.