Rechnerarchitekturen 1*

Exceptions & Interrupts

Prof. Dr. Alexander Auch

*Teilweise entnommen aus “Mikrocomputercomputertechnik 1” von Prof.Dr-Ing. Ralf Stiehler, sowie Patterson & Hennessy

Ziele der Veranstaltung

- **Rechnerentwurf**:
  - Prozessor, Speicher, Ein-/Ausgabe
  - Entwurfs- und Optimierungsmöglichkeiten
- **Prozessorentwurf**:
  - Befehlsverarbeitung
  - Entwurfs- und Optimierungsmöglichkeiten
- **Assemblerprogrammierung**:
  - im MIPS-Simulator MARS

Mikrocomputertechnik 1 – Exception Handling

Behandlung von Ausnahmen (exception handling)

Unter Ausnahmen (exception) versteht man unerwartete Ereignisse, die den normalen Kontrollfluss eines Programms verändern.

Exceptions können asynchron und synchron auftreten

Asynchrone Ereignisse werden als Interrupts bezeichnet.
Sie werden durch äußere Ereignisse ausgelöst, z.B. I/O Interrupts,…

Synchrone Ereignisse bezeichnet man als Traps. Ursache sind interne Ereignisse (z.B. Division durch 0, Overflow oder ungültiger Befehl)

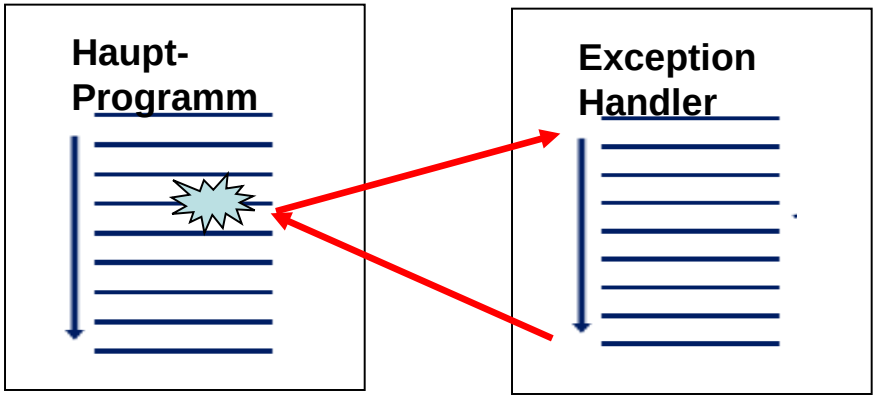
Die Steuerung muss so geändert werden, dass sie auf diese Exceptions entsprechend reagieren kann

Die Verarbeitung von Exceptions erfolgt in 3 Schritten

1) Befehlszählerstand feststellen, bei dem die Exception auftritt
2) Ursache der Exception feststellen
3) Übergabe der Steuerung an den Exception Handler (i.d.R. das Betriebssystem)

Mikrocomputertechnik 1 – Exception Handling

Aufgaben des Exception Handlers



- Vordefinierte Aktionen durchführen
  z.B. (Reaktion auf Overflow)
- ggf. Hauptprogramm stoppen und
  Fehlermeldung ausgeben
- ggf. Benutzerdaten sichern und
  wiederherstellen (ähnlich wie bei Unterprogramm – Stackrettung)
- Rückgabe der Kontrolle ans Hauptprogramm nach Abarbeitung der Exception

Ursache der Exception feststellen und speichern

Möglichkeit 1: Status Register - Single entry point for all exceptions

Die Ursache wird in einem speziellen Statusregister, dem Cause Register codiert

Möglichkeit 2: Vectored interrupt

Die Adresse zu der verzweigt wird, ist durch die Ursache der Exception festgelegt. An dieser Adresse steht dann die entsprechende Routine, die dann in der Regel zur eigentlichen Exception-Routine weiterverlinkt. Der Exception-Handler kennt den Exception-Grund nur aufgrund der Adresse wohin verzweigt wurde.

Implementierung der Exceptions

Das Erkennen und das Behandeln von Exceptions gehört in der Regel zu den schwierigsten Aufgaben beim Steuerwerksentwurf.

Das Exception-Handing bestimmt oft den kritischen Pfad im Prozessor und bestimmt die maximale Taktfrequenz.

Wir werden beispielhaft 2 Exceptions im Steuerwerk implementieren.
- ungültige Instruktion durch ungültigen OP-Code
- arithmetrischer Overflow

Implementierung erfolgt mit der erstgenannten Methode, einem Statusregister

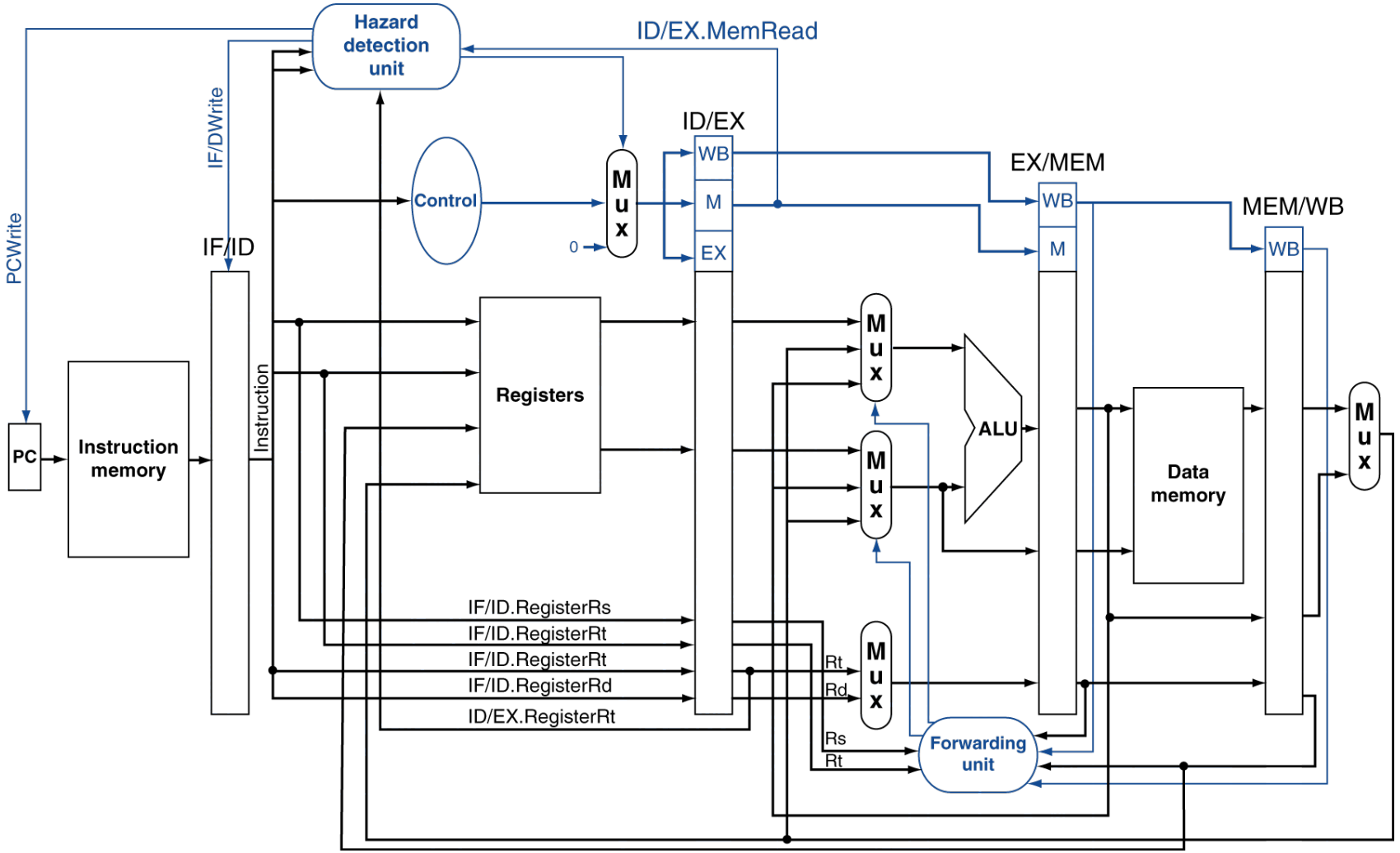
Es wird zusätzliche Hardware benötigt :

- EPC (exception program counter)
  32 Bit Register zur Speicherung der Adresse, bei dem die Exception auftrat
- Cause Register : Codierung der Ursache 1(ungültiger Befehl) 2 (arithmetr. Overflow)
- erweiterter PCSRC MUX , um die Exception Handling Adresse in den PC zu laden

Die Exception-Handling-Adresse ist bei MIPS festgelegt auf 0x 8000 0180.

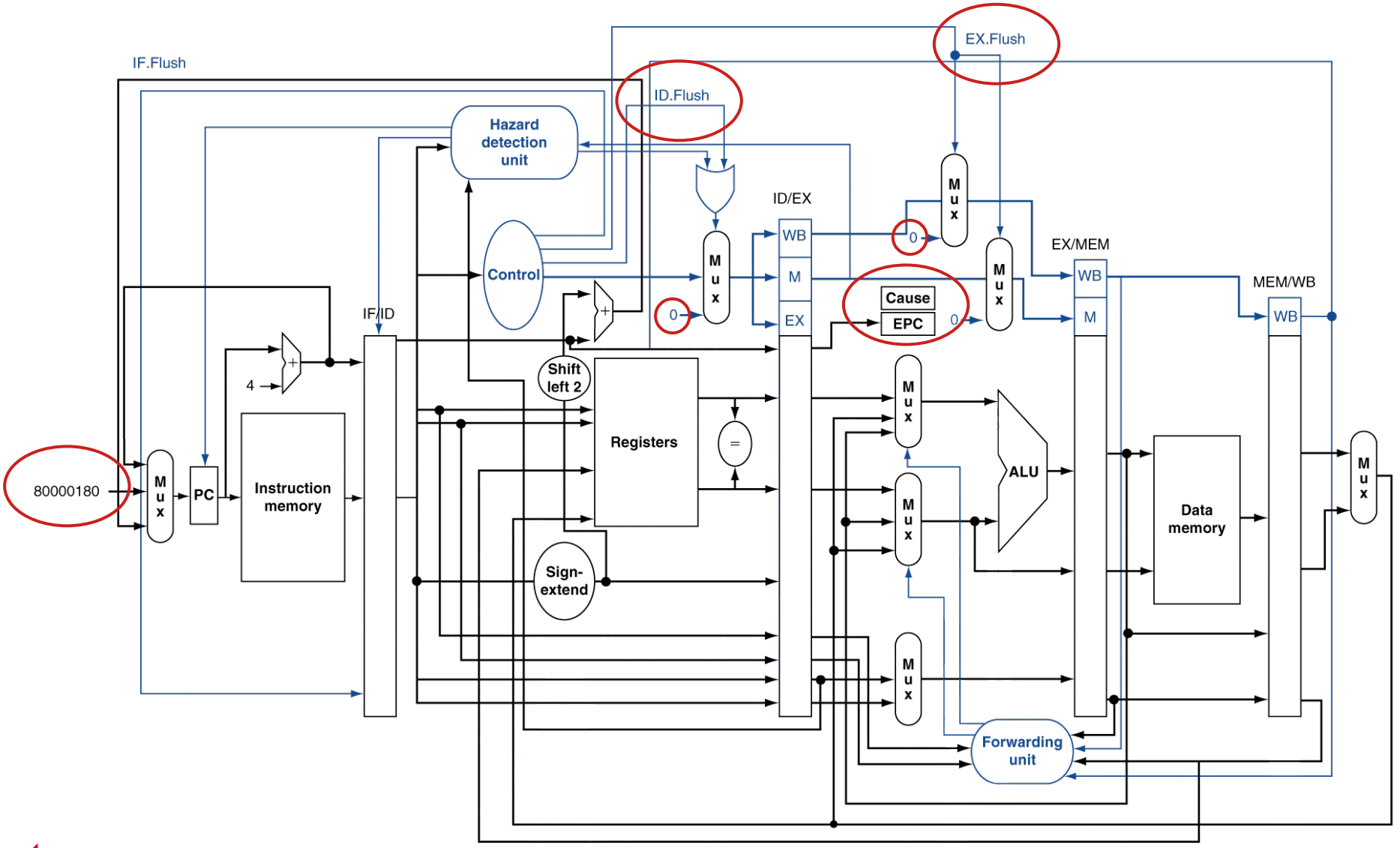
Mikrocomputertechnik 1 – Exception Handling

Pipeline-Datenpfad ohne Exception-Logik



Mikrocomputertechnik 1 – Exception Handling

Erweiterung des Pipeline-Datenpfads



Hinweise zu den Erweiterungen

Die Pipeline behandelt diese wie ein Control-Hazard → eine Verzweigung.
Alle späteren Anweisungen in der Pipeline müssen zu NOPs werden (Steuerleitungen)

Es gibt eine Exception Handling Adresse, die als Single Entry Point für Ausnahmen angesprungen wird : 0x 8000 0180

Der Befehlszähler wird beim normalen Multicycle bereits im ersten Zyklus immer um 4 erhöht. Deswegen muss bei einer Exception erst 4 subtrahiert werden, bevor der Befehlszählerstand in das EPC Register geschrieben wird.

Ablauf einer Exception

⇒ Bei einer Ausnahme wird die Ursache der Exception codiert und mit der nächsten Taktflanke ins Cause-Register geschrieben.
⇒ PC-4 wird berechnet und in das Register EPC geschrieben
⇒ Der Befehlszähler wird mit der Adresse 0x 8000 0180 beschrieben
⇒ Ab Adresse 0x8000 0180 ist die Exception Routine, die zunächst das Cause-Register auswertet und dann weitere Aktionen durchführt.

Der Kernel

⇒ Der **Kernel** ist der Kern des Betriebssystems und verarbeitet die eben gerade behandelten Exceptions (auch Interrupts und System Calls => kommt noch).

⇒ Der **Kernel** ist ein Programm und besteht aus Programmcode und Daten wie jedes andere Programm auch.

⇒ **Kernel**-Programmcode arbeitet in einem sog. privilegierten Modus, d.h. er hat Zugriff auf den gesamten Befehlssatz und den gesamten Speicherbereich

Die **Kernel-Privilegien** umfassen :

⇒ Ansteuerung von Hardware

⇒ Isolierung von anderen Prozessen, d.h. kein Programm kann Kernelprogramm-code lesen, ändern oder ausführen (Ausnahme sind kontrollierte Abläufe, z.B. Syscalls)

⇒ Ausführung spezieller Befehle (wie etwa halt)

Kernel, Betriebssystem und Memory Map

⇒Der **Kernel-**Programmcode liegt in einem bestimmten Speicherbereich des externen Speichers.

⇒Auch Anwenderprogramme, und Anwenderdaten liegen im externen Speicher in einem jeweils **definiertem Speicherbereich**

⇒Definition einer **Memory Map**

⇒Das **Betriebssystem** umfasst neben dem Kernel weitere Programme wie etwa Hilfsprogramme (z.B. Lesen einer Datei), Befehlsinterpreter, Libraries o.ä. Diese stehen den Anwendungsprogrammen als Schnittstelle zur Verfügung.

MEMORY MAP: MIPS Speicherkonvention

Stacksegment

⇒Sicherung von Registerinhalten bei Prozeduraufrufen adressiert über Register $sp (stack pointer)

Datensegment

⇒statische Daten i.d.R. adressiert über Register $gp (global pointer)

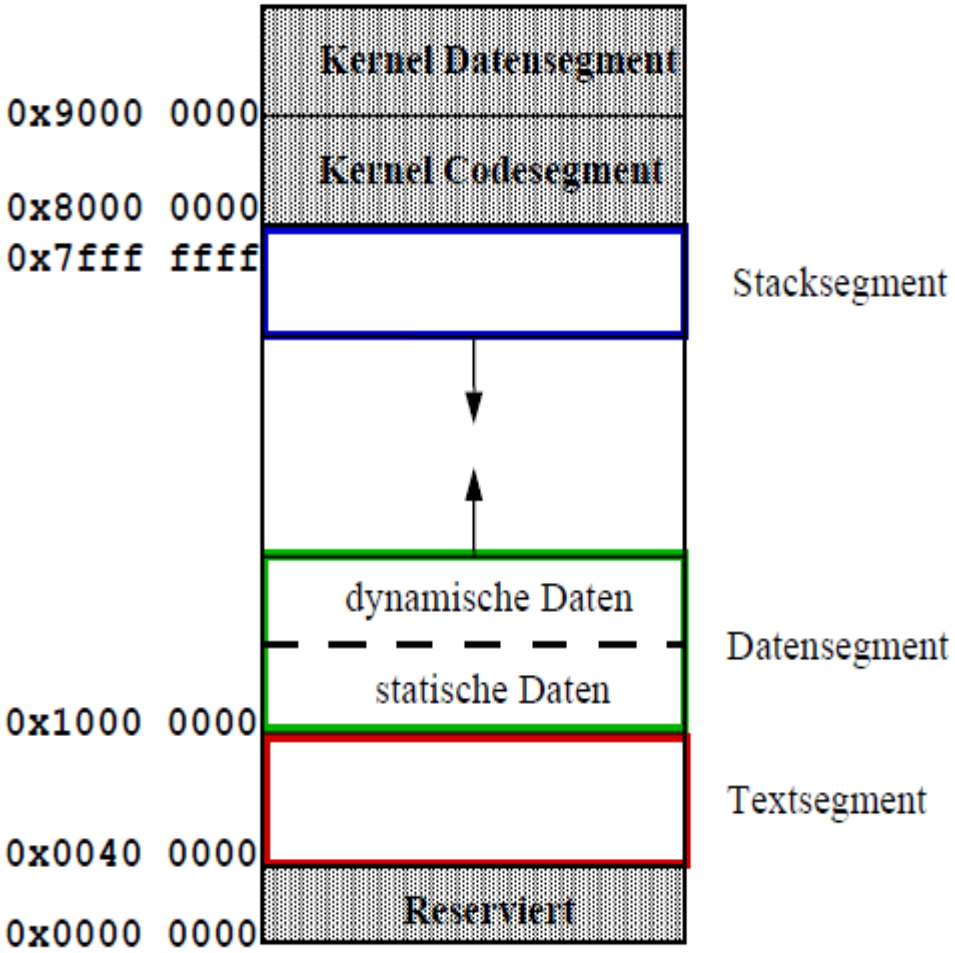
⇒dynamische Daten

Textsegment

⇒auszuführende Befehlsfolge adressiert über Program Counter

reservierter Bereich

⇒Betriebssystemcode (Kernel)



Betriebssystemaufrufe

⇒Anwenderprogramme, die direkt auf die Hardware zugreifen, wären viel zu kompliziert, z.B. erfordert das Ändern des Graphikmodus >20 Befehle

⇒Hardwareansteuerung über **Syscall (System Call oder Betriebssystemaufruf**)

⇒System Calls sind die Schnittstelle zwischen Prozessen und Kernel.

⇒Ein Prozess verwendet Syscalls um Dienstleistungen vom Betriebssystem zu erhalten, z.B. Dateien lesen / schreiben oder Bildschirmausgabe von Zeichen

Hardwareansteuerung über Syscall

1. Im Register $v0 wird die vom Betriebssystem auszuführende Funktion hinterlegt
2. Über weitere Register ($a0-$a3) werden die Funktionsargumente übergeben
3. Befehl SYSCALL ausführen
4. Evtl. Rückgaben des Kernels/Betriebssystems erfolgen über $v0

Syscall-Beispiel 1 : **Ausgabe eines Integerwerts, der im Register $t2 abgelegt ist**

```
addi $v0,$zero, 1      # Laden des Syscall-Codes 1 ins Register $v0
                       # Syscall-Code 1 steht für Integerwert drucken
add $a0,$zero $t2      # Integerwert von $t2 nach $a0 kopieren
Syscall                # Aufruf des Kernals/Betriebssystems,  um diese
                       # Hardwareoperation durchzuführen
```

Syscall-Beispiel 2 : **Einlesen eines Integerwerts von der Tastatur**

```
addi $v0,$zero, 5      # Laden des Syscall-Codes 5 ins Register $v0
                       # Syscall-Code 5 steht für Integerwert einlesen
Syscall                # Aufruf des Kernels/Betriebssystems
```

Verfügbare Syscalls bei MIPS

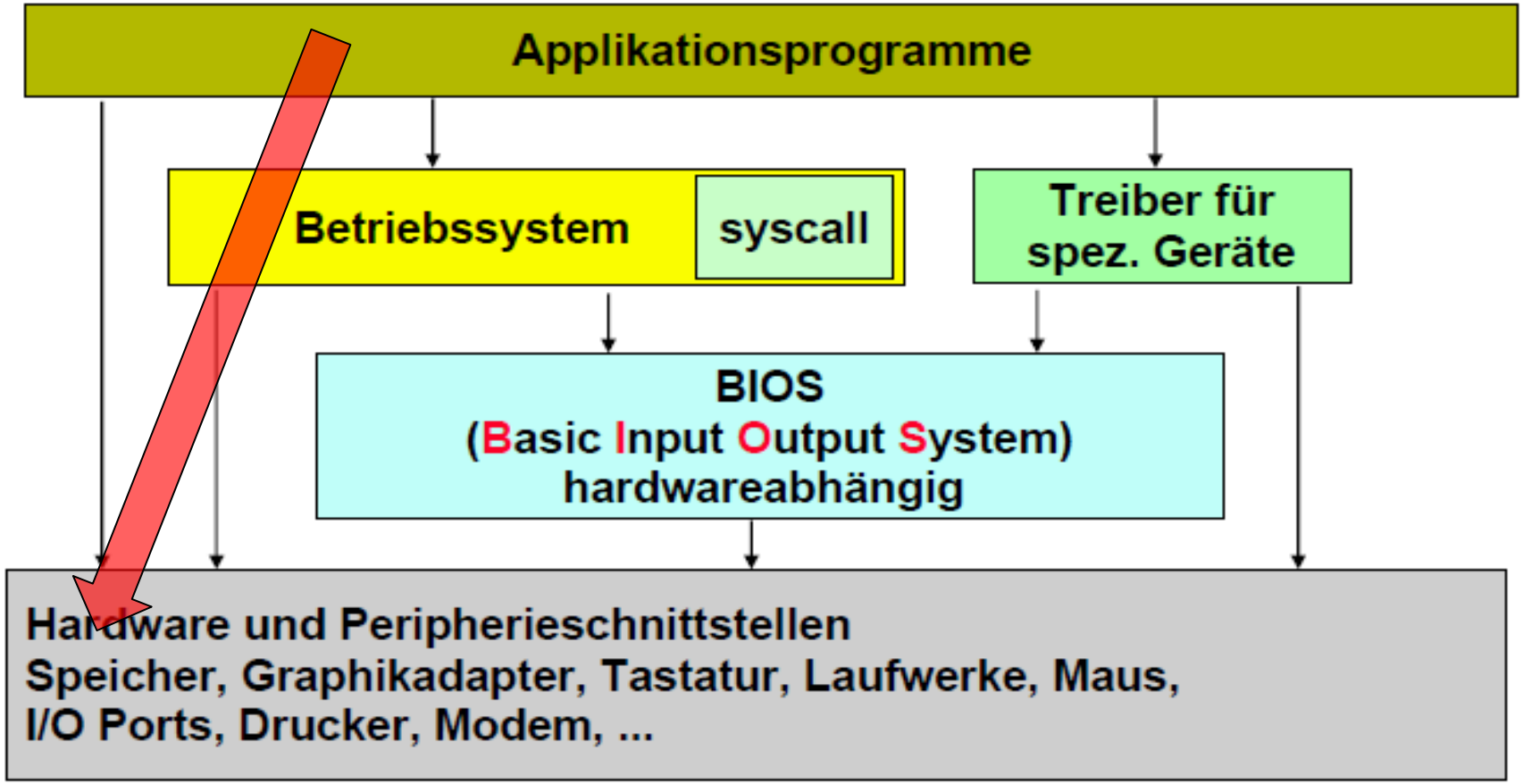
Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $v0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

Ablauf eines Betriebssystemaufrufs (Syscalls)

(Ablauf analog zum Exception- oder Interrupt-Handling)

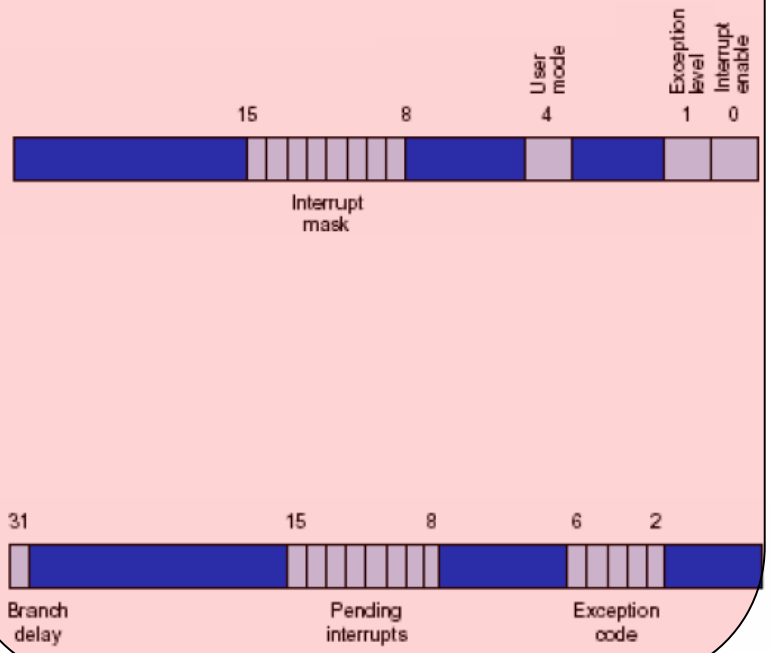
⇒ Prozessor wird auf privilegierten Modus umgeschaltet
⇒ Registerinhalte und Befehlszählerstand des aktuellen Programms werden in den externen Speicher gerettet (über Stack)
⇒ Befehlszähler wird mit einer Speicheradresse des Kernel geladen.
⇒ An dieser Speicheradresse liegt ein Programm, der System Call Handler
⇒ Handler stellt zu erledigende Aufgabe anhand des Syscodes fest, der über Register $v0 bzw. $a0 übergeben wurde
⇒ Aufgabe wird erledigt , d.h. Ausführung eines Programms im Kernel)
⇒ Zurückschalten des Prozessor auf nicht-privilegierten Modus
⇒ Ursprüngliche Registerinhalte werden über den Stack wiederhergestellt
⇒ Befehlszählerstand wird wiederhergestellt
⇒ Anwenderprogramm läuft normal weiter

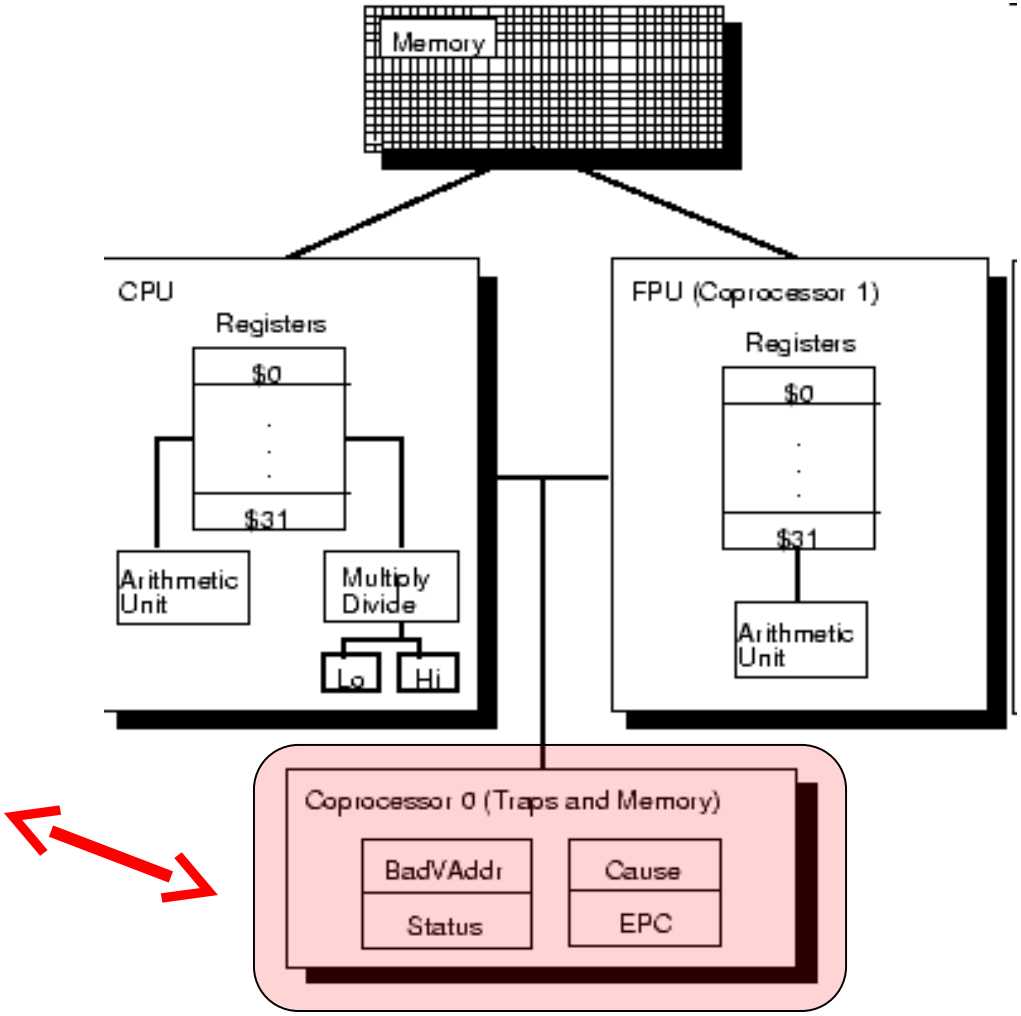
Weitere Methoden zur Hardwareansteuerung



Mikrocomputertechnik 1 – Exception Handling

„Coprocessor 0“





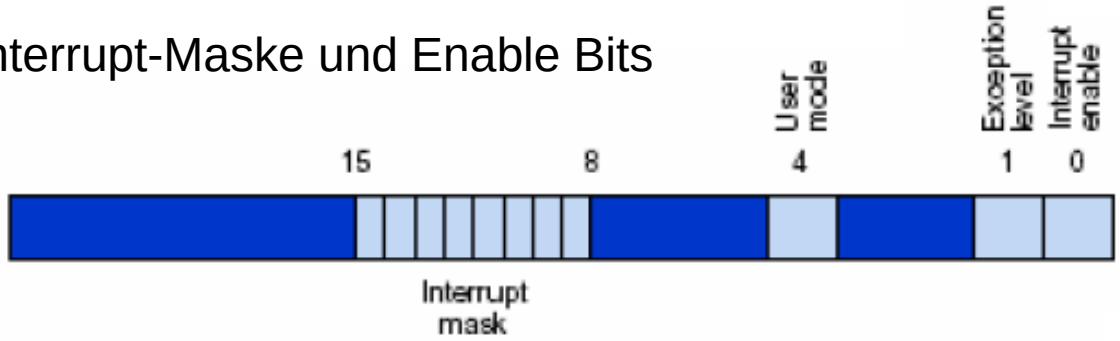
Exception Handler bei MIPS/MARS

⇒Der Exception Handler greift auf „Coprocessor 0“ zu (Anordnung von Registern)

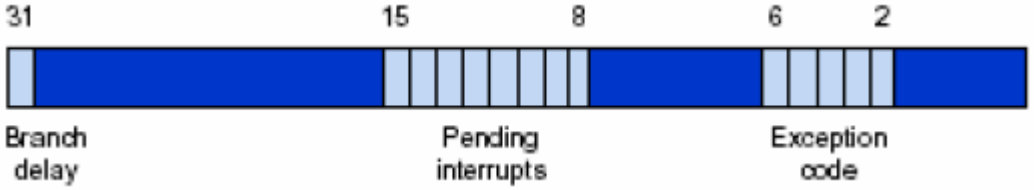
⇒ EPC ($14): Hier wird die Instruktionsadresse gespeichert,
die die Exception auslöste bzw. bei der ein Interrupt auftrat

⇒BadVAddr ($8): Speicheradresse eines fehlschlagenden Speicherzugriffs

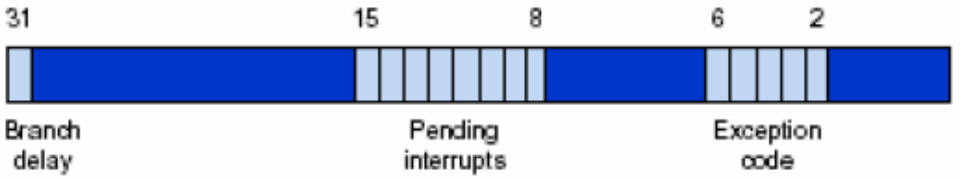
⇒Status ($12): Interrupt-Maske und Enable Bits



⇒Cause ($13) : Hier ist die Ursache des Interrupts oder der Exception hinterlegt:



MIPS/MARS Exception Codes



Nummer	Exception-Typ (Grund)
0	Interrupt (Hardware)
4	Address-Error (load or fetch)
5	Address-Error (store)
6	Bus-Error (fetch)
7	Bus-Error (store)
8	System-Call
9	Break-Point
10	Reserved Instruction
11	Coprocessor Unimplemented
12	Arithmetic Overflow
13	Trap
15	Floating-Point-Exception

Befehle zur Arbeit mit dem Coprozessor / Exceptions:

Move from Coprocessor 0

```
mfc0 $t0, $13
```

Move to Coprocessor 0

```
mtc0 $t0, $14
```

Trap if equal (löst Exception 13 aus)

```
teq $t0, $1
```

Return from Exception to EPC

```
eret
```

Interrupts

⇒ Neben Exceptions und Syscalls sind Interrupts ein weiterer Mechanismus, um Kontrolle der CPU an das Betriebssystem (Kernel) zu übergeben

⇒ Interrupts sind sehr ähnlich zu synchronen Traps, nur werden diese asynchron durch Hardware ausgelöst und treten nicht im normalen Programmablauf auf

Beispiele für Interrupts :

⇒ ein Netzwerkcontroller löst einen Interrupt aus, wenn ein Datenpaket gekommen ist
⇒ ein Timer ist abgelaufen und zeigt dies durch einen Interrupt an

Das **Interrupt-Handling** funktioniert ganz analog zum Exception-Handling
⇒ CPU-Register und aktuelle Programmausführungsadresse auf Stack retten
⇒ Codierung der Interrupt-Ursache in einem internen Register (Cause Register)
⇒ Übergabe der Steuerung an das Kernel, das die Interruptroutine abarbeitet
⇒ Wiederherstellen der CPU-Register und Zurückspringen ins Hauptprogramm

Demo eines einfachen Exception Handlers im MARS-Simulator

Exception auslösendes Hauptrogramm:
`exception_testprogramm.asm`

Exception-Handler :
`exception-file.asm`

Der beispielhafte Exception-Handler fängt nur die im Beispielprogramm `exception_testprogramm.asm` auftretende Exception ab (erzwungener Trap durch trap_if_equal-Befehl `TEQ` , Codierung Nr. 13)

Ihre Aufgabe(n) :

a) Exception-Code anschauen und nachvollziehen
b) Fehlerbehandlung für unbekannte Exceptions abändern, sodass das Programm beendet wird (via syscall)
c) Implementieren einer Fehlerbehandlungsroutine für 2k-Overflow
d) Implementieren einer Fehlerbehandlungsroutine für sw / lw-Speicherfehler