Hochschule München University of Applied Sciences

IT-SYSTEME

RISC-V Privileged Architecture / Exception / Interrupt

Prof. Dr. Martin Hobelsberger, Prof. Dr. Martin Orehek, Prof. Dr. Stefan Wallentowitz

Fakultät für Informatik und Mathematik

RV32/64 Privileged Architecture

- Bis jetzt haben wird die RISC-V *Instruction Set Achitecture* (ISA) für den *User Mode* kennen gelernt
- Jetzt werden wir zwei neue, höher privilegierte Modes kennen lernen
  - *Machine Mode* in dem der Code mit dem größten Vertrauen im System läuft
  - *Supervisor Mode* zur Umsetzung von Betriebssystemen (z.B. Linux, FreeBSD, Windows, ...)
- Was bedeutet "höher privilegiert"?
  - in diesen Modes kann alles gemacht werden was auch in den darunter liegenden, weniger privilegierten Modes gemacht werden kann
  - es werden neue weiterführende Funktionen hinzugefügt, die nur im privilegierten Mode möglich sind. Z.B. das Bedienen von Interrupts oder die Umsetzung von Peripheriezugriffen

RISC-V Volume II: Privileged Architecture

The RISC-V Instruction Set Manual
Volume II: Privileged Architecture
Document Version 20190608-Priv-MSU-Ratified

Editors: Andrew Waterman[1], Krste Asanović[1,2]
[1]SiFive Inc.,
[2]CS Division, EECS Department, University of California, Berkeley
andrew@sifive.com, krste@berkeley.edu
June 8, 2019

RISC-V Volume II: Privilege Levels

Level	Encoding	Name	Abbreviation
0	00	User/Application	U
1	01	Supervisor	S
2	10	*Reserved*	
3	11	Machine	M

Table 1.1: RISC-V privilege levels.

- *Machine Mode (M-mode)* ... höchsten Privilegien, da low-level Zugriff auf die HW erlaubt ist. Der Code in diesem Mode sollte sicher sein!
- *User Mode (U-mode)* ... niederste Ebene der Privilegien, für gewöhnlich laufen hier die Applikationen. In diesem Zustand wird das System die meiste Zeit sein.
- *Supervisor Mode (S-mode)* ... das ist für gewöhnlich die Ebene des Betriebssystems. Liegt zwischen Machine und User Mode.

RV32/64 Privileged Architecture

- Üblicherweise verbringt ein System die meiste Zeit im am wenigsten privilegierten Mode, erst Interrupts oder Exceptions führen zu einen Übergang in höher privilegierte Modes
- Was bring also diese Privileged Architektur für Erweiterungen?
  - nur wenige neue Befehle (siehe nächsten Abschnitt)
  - einige neue Control und Status Register (CSRs)

RV32/64 Privileged Instructions

- Die RV Privilegted Instructions sind
  - `MRET` ... Machine-Mode Trap Return
  - `SRET` ... Supervisor-Mode Trap Return
  - `SFENCE.VMA` ... Supervisor-Mode Fence Virtual Memory Address
  - `WFI` ... Wait for Interrupt

Machine Mode

- Ist der Mode mit den meisten Privilegien
  - Voller Zugriff auf den Speicher
  - Voller Zugriff auf I/O
  - Voller Zugriff auf low-level System Feature, die zum Booten (Starten) und Konfigurieren des Systems notwendig sind
- Es ist somit der einzige Mode, der von allen RISC-V Prozessoren implementiert werden muss
- Die wichtigste Eigenschaft dieses Modes ist die Fähigkeit Exceptions und Interrupts abzufangen und zu bedienen

Machine Mode Exception Handling

- Es gibt 8 Control und Status Register (CSRs) hierzu
- `mstatus` = *Machine Status* ... beinhaltet Informationen zum aktuellen Zustand eines Hardware Threads und steuert diesen auch (z.B. Interrupt Enable Flags)
- `mip` = *Machine Interrupt Pending* ... gibt an welche Interrupt Requests gerade warten
- `mie` = *Machine Interrupt Enable* ... gibt an welche Interrutps von dem Prozessor angenommen werden und welche ignoriert werden
- `mtvec` = *Machine Trap-Vector Base-Address* ... hält die Basis Adresse des Trap-Handlers

Machine Mode Exception Handling

- `mcause` = *Machine Cause* ... gibt an welches Ereignis den aktuellen Trap ausgelöst hat
- `mtval` = *Machine Trap Value* ... enthält entweder 0 oder zusätzliche Trap Informationen (z.B. Fehlerhafte Adresse beim Speicherzugriff, OpCode der Illegalen Instruktion, ...)
- `mepc` = *Machine Exception Program Counter* ... Adresse der Instruction die Unterbrochen wurde oder bei der die Exeption aufgetreten ist
- `mscratch` = *Machine Scratch* ... stellt ein Element von Daten zur temporären Sicherung bereit, oft wird ein Zeiger auf einen Speicherbereich zur lokalen Sicherung des Contextes hinterlegt

Machine Status

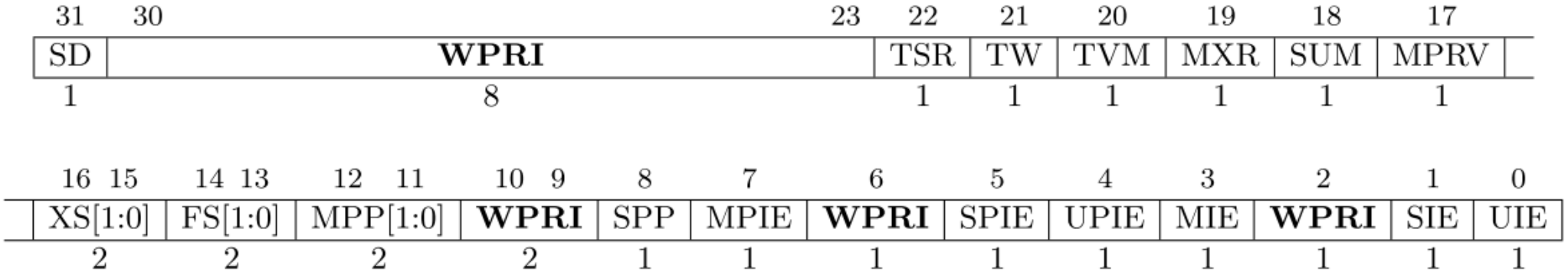


Figure 3.6: Machine-mode status register (`mstatus`) for RV32.

WPRI

Reserved Writes Preserve Values, Reads Ignore Values (WPRI)

Some whole read/write fields are reserved for future use. Software should ignore the values read from these fields, and should preserve the values held in these fields when writing values to other fields of the same register. For forward compatibility, implementations that do not furnish these fields must hardwire them to zero. These fields are labeled **WPRI** in the register descriptions.

Machine Status Details

- Globale Interrupt Enable-Bits für die unterschiedlichen Modi: `MIE`, `SIE`, `UIE`
- `mstatus` hält den sogenannten *two-level Privilege and Global Interrupt-Enable Stack* (max. 1 Verschachtelung)
- D.h. um Nested-Traps zu ermöglichen gibt es für jeden Privilege-Mode einen *two-level stack*
  - `xPIE` *previous interrupt enable mode* und
  - `xPP` *previous privilege mode*.
- Beim Eintritt in einen Privilege-Mode werden die Interrupts deaktiviert und der Privilege-Zielmode eingenommen. Gleichzeitig wird die aktuelle Konfiguration des Interrupt Enable Flags und das Privilege-Level des Ausgangszusatndes gespeichert

Rückkehr aus Trap Handler / Wiederherstellung des Zustandes

- Beim Rückkehren aus dem Trap werden bei den Befehlen `MRET`, `SRET` und `URET` die *previous* Werte in einer atomaren Operation auf die eigentlichen Bits zurückgeschrieben
- Damit wird der vorhergehende Zustand bestehen aus *Interrupt Enable* + *Privileged Mode* wieder hergestellt
- Konkret bedeutet das, dass beim Ausführen von `xRET`
  - der Inhalt von `xPP` den *priviledge mode* nach der Ausführung definiert
  - `xPIE` den `xIE` Wert nach der Ausführung definiert
  - nach der Ausführung wird `xPIE` = 1 gesetzt und `xPP` auf `U` (oder `M` falls *user-mode* nicht unterstützt wird)

Machine Status Frage

- Warum gibt es konkret folgende Bitbreiten?
  - MPP 2 Bit
  - SPP 1 Bit
  - UPP 0 Bit (nicht vorhanden, implizit immer 0)

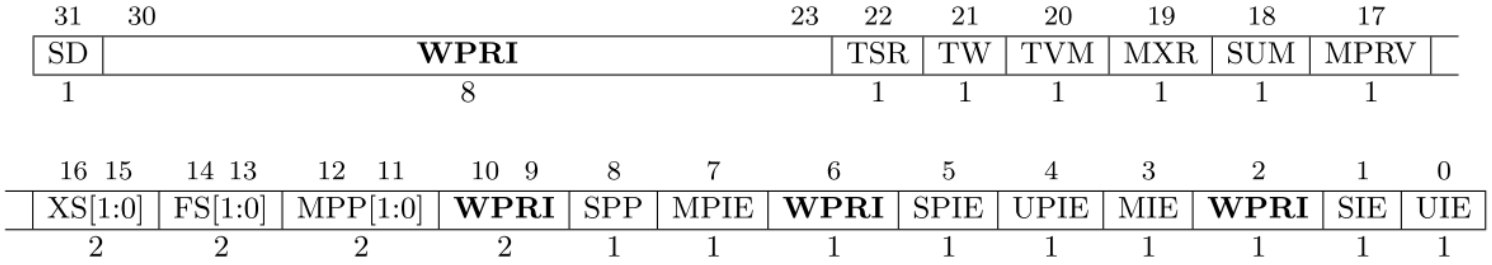


Figure 3.6: Machine-mode status register (`mstatus`) for RV32.

Machine Interrupt Pending / Machine Interrupt Enable

MXLEN-1 … 12	11	10	9	8	7	6	5	4	3	2	1	0
WPRI	MEIP	**WPRI**	SEIP	UEIP	MTIP	**WPRI**	STIP	UTIP	MSIP	**WPRI**	SSIP	USIP
MXLEN-12	1	1	1	1	1	1	1	1	1	1	1	1

Figure 3.11: Machine interrupt-pending register (`mip`).

MXLEN-1 … 12	11	10	9	8	7	6	5	4	3	2	1	0
WPRI	MEIE	**WPRI**	SEIE	UEIE	MTIE	**WPRI**	STIE	UTIE	MSIE	**WPRI**	SSIE	USIE
MXLEN-12	1	1	1	1	1	1	1	1	1	1	1	1

Figure 3.12: Machine interrupt-enable register (`mie`).

Machine Interrupt Pending / Machine Interrupt Enable Details

- Status / Steuerung asynchroner Unterbrechungen == "externe" Interrupts
- `mip` gibt an welche Interrupts *pending* sind, also auf Bearbeitung warten
  - Low-level Software Interrupts: `USIP`, `SSIP` (z.B. Inter-Prozessor Kommunikation)
  - Timer Interrupts: `UTIP`, `STIP` (siehe `mtimer` bzw. `mtimercmp`)
  - External Interrupts: `UEIP`, `SEIP`
- `mie` analog dazu gibt es die Enalbe Interrpupt Bits

Machine Trap-Vector Base-Address

Figure 3.8: Machine trap-vector base-address register (`mtvec`).

Machine Trap-Vector Base-Address Details

- BASE stellt die Adresse des Trap-Handlers dar (*Direct Mode*) bzw. die Start-Adresse der Handler-Tabelle (*Vectored Mode*)

Value	Name	Description
0	Direct	All exceptions set `pc` to BASE.
1	Vectored	Asynchronous interrupts set `pc` to BASE+4×cause.
≥2	—	*Reserved*

Table 3.5: Encoding of `mtvec` MODE field.

Machine Cause

- Aus der Spezifikation:
  - *Interrupts can be separated from other traps with a single branch on the sign of the mcause register value.*
  - *A shift left can remove the interrupt bit and scale the exception codes to index into a trap vector table.*

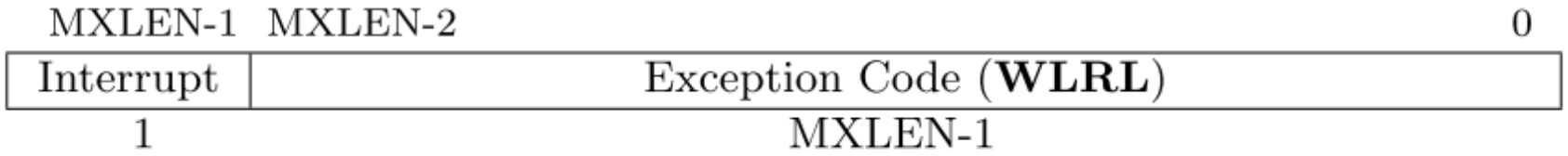


Figure 3.22: Machine Cause register `mcause`.

Machine Cause Details

Interrupt	Exception Code	Description
1	0	User software interrupt
1	1	Supervisor software interrupt
1	2	*Reserved for future standard use*
1	3	Machine software interrupt
1	4	User timer interrupt
1	5	Supervisor timer interrupt
1	6	*Reserved for future standard use*
1	7	Machine timer interrupt
1	8	User external interrupt
1	9	Supervisor external interrupt
1	10	*Reserved for future standard use*
1	11	Machine external interrupt
1	12–15	*Reserved for future standard use*
1	≥16	*Reserved for platform use*
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	*Reserved*
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	*Reserved for future standard use*
0	15	Store/AMO page fault
0	16–23	*Reserved for future standard use*
0	24–31	*Reserved for custom use*
0	32–47	*Reserved for future standard use*
0	48–63	*Reserved for custom use*
0	≥64	*Reserved for future standard use*

Machine Cause Details Interrupts

Interrupt	Exception Code	Description
1	0	User software interrupt
1	1	Supervisor software interrupt
1	2	*Reserved for future standard use*
1	3	Machine software interrupt
1	4	User timer interrupt
1	5	Supervisor timer interrupt
1	6	*Reserved for future standard use*
1	7	Machine timer interrupt
1	8	User external interrupt
1	9	Supervisor external interrupt
1	10	*Reserved for future standard use*
1	11	Machine external interrupt
1	12–15	*Reserved for future standard use*
1	≥16	*Reserved for platform use*
0	0	Instruction address misaligned

Machine Cause Details Exceptions

1	≥16	*Reserved for platform use*
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	*Reserved*
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	*Reserved for future standard use*
0	15	Store/AMO page fault
0	16–23	*Reserved for future standard use*
0	24–31	*Reserved for custom use*
0	32–47	*Reserved for future standard use*
0	48–63	*Reserved for custom use*
0	≥64	*Reserved for future standard use*

Machine Trap Value

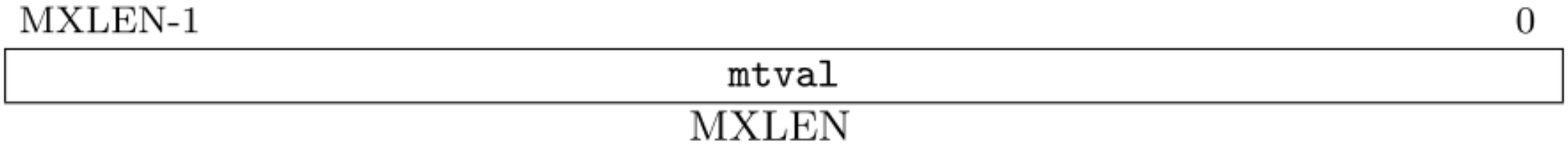


Figure 3.23: Machine Trap Value register.

Machine Exception Program Counter

- Aus der Spezifikation: *When a trap is taken into M-mode, mepc is written with the virtual address of the instruction that was interrupted or that encountered the exception.*

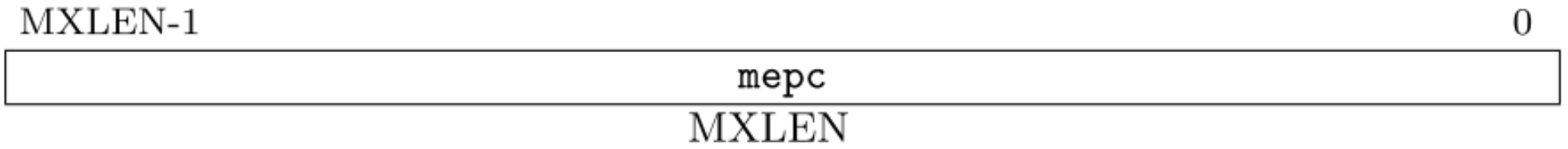


Figure 3.21: Machine exception program counter register.

Machine Scratch

Figure 3.20: Machine-mode scratch register.

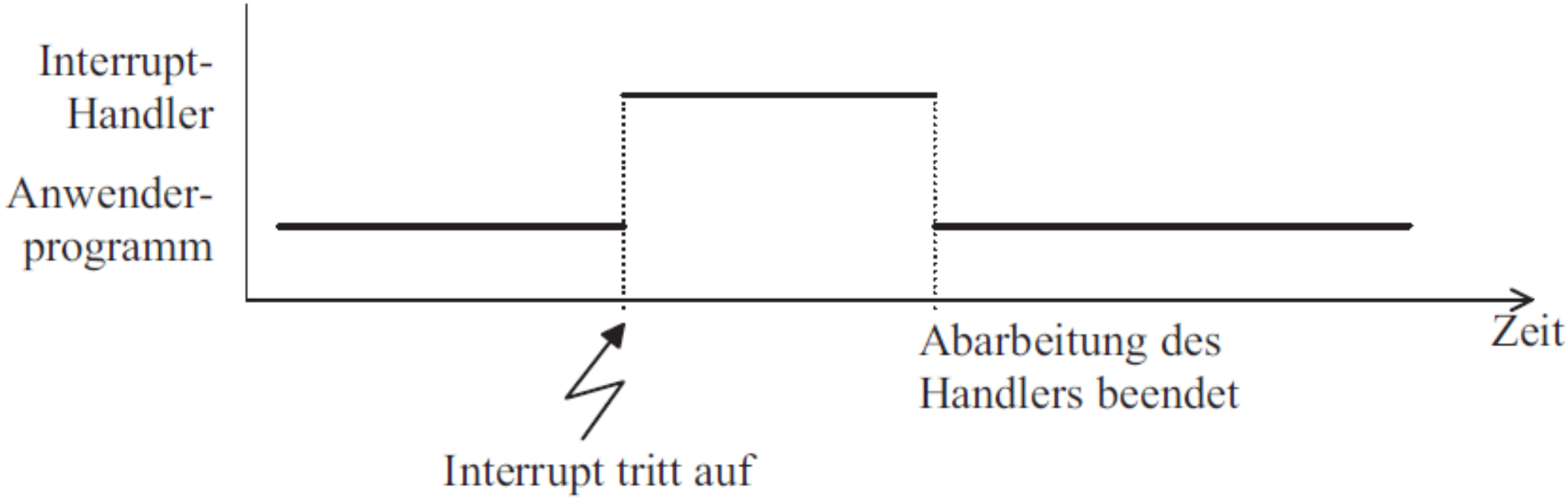
Machine-Mode Priviledged Instructions

- `ecall`
  - Spezifische Funktions-Anfrage an das *Execution Environment*
  - Kann von *user-mode* (U-mode), *supervisor-mode* (S-mode) oder *machine-mode* (M-mode) aus erfolgen
  - Löst jeweils eine unterschiedliche Exception (synchrone Unterbrechung) aus
- `ebreak`
  - Nutzen Debugger, um die Programm-Kontrolle an die Debug-Umgebung zu übergeben (z.B. ein gesetzter Breakpoint)
- `xRET`

xRET

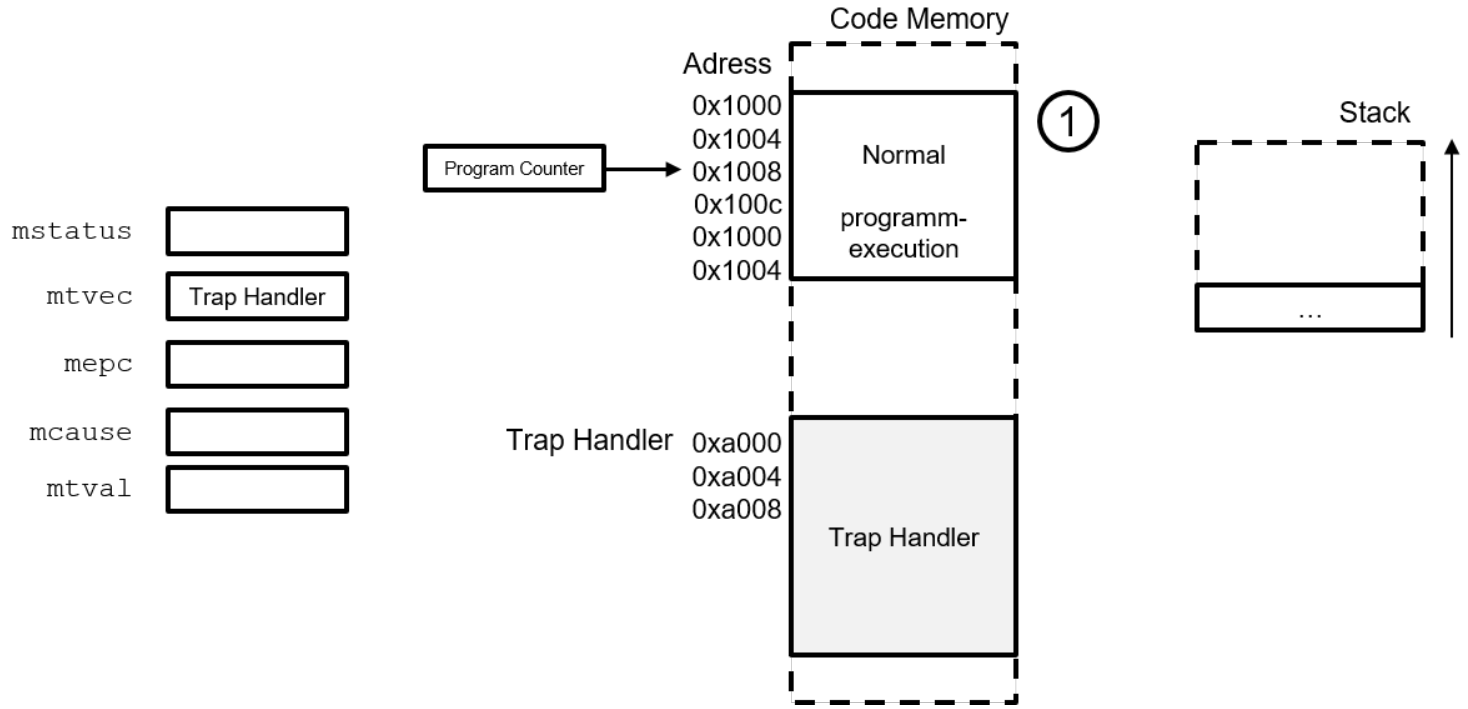
- Trap-Return Instruktionen
- Für jedes Privilege-Level gibt es einen eigenen Befehl
- Das System muss aktuell im gleichen oder höheren Privilege-Level sein um den Befehl auszuführen
- Folgende Aktionen werden ausgelöst
  - Wiederherstellen des Interrupt Enable Zustands und Privilege-Levels, siehe Details bei `mstatus` vorher
  - Program Counter wird mit dem Wert aus `xepc` überschrieben, diese entspricht einem Rücksprung zum unterbrochenen Programm

Zeitlicher Ablauf der Trap Verarbeitung RISC-V



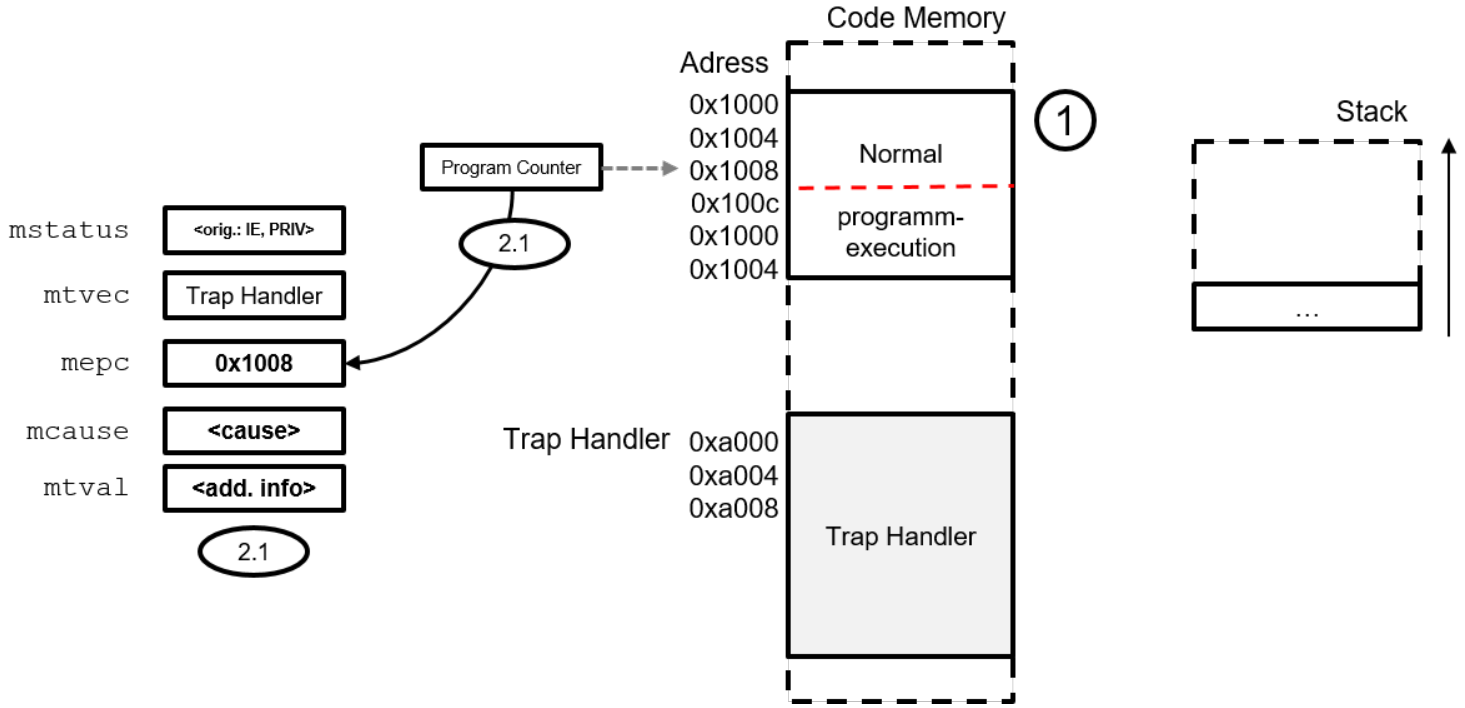
Trap Verarbeitung RISC-V

1. Die CPU arbeitet ein Programm ab



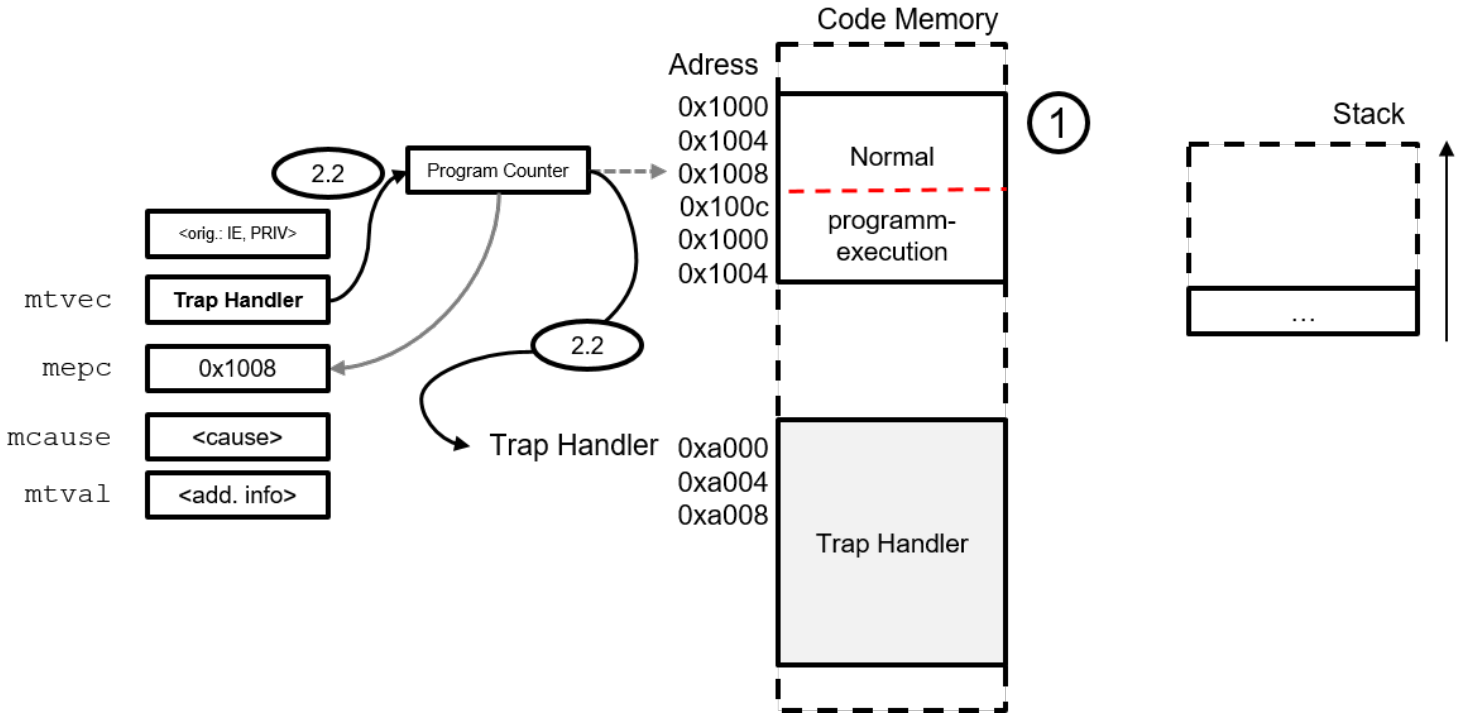
Trap Verarbeitung RISC-V

1. Die CPU arbeitet ein Programm ab
2. Interrupt während der Programmabar-beitung



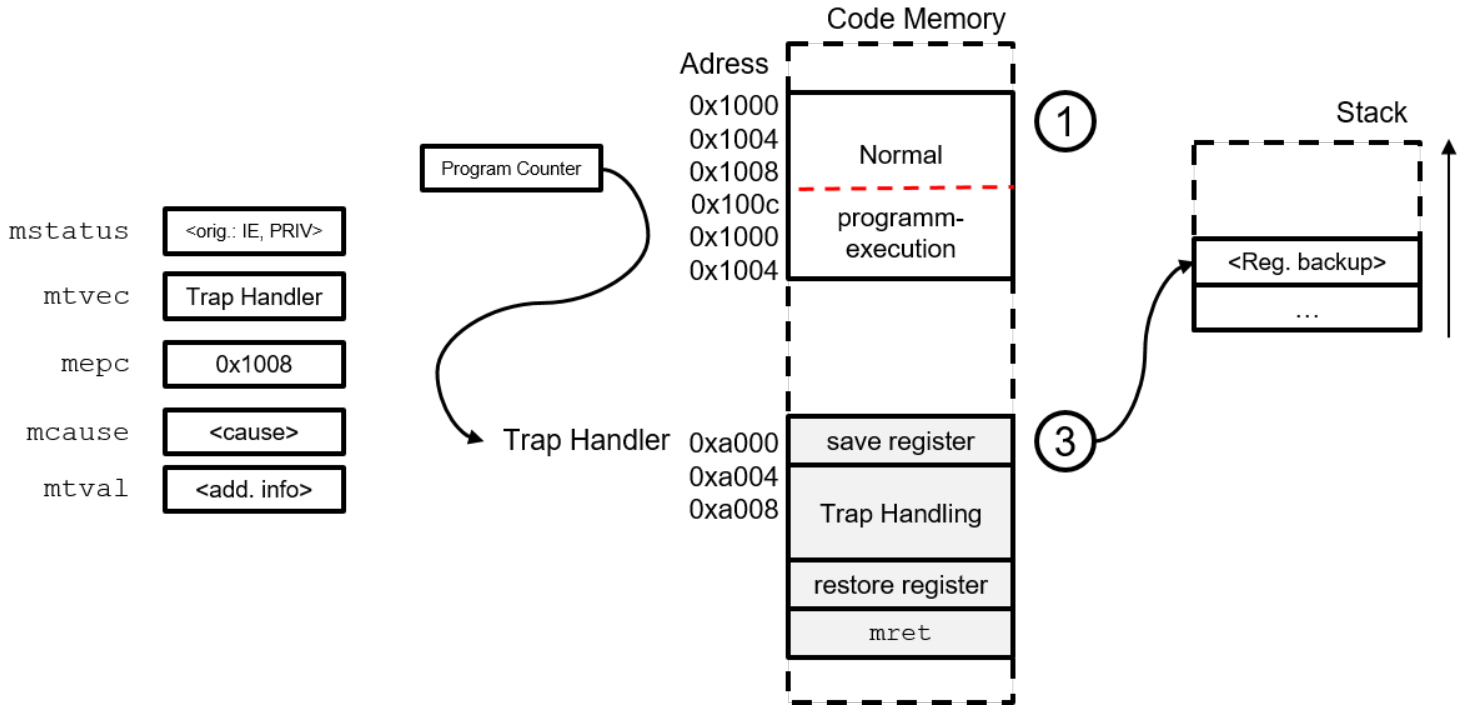
Trap Verarbeitung RISC-V

1. Die CPU arbeitet ein Programm ab
2. Interrupt während der Programmabar-beitung



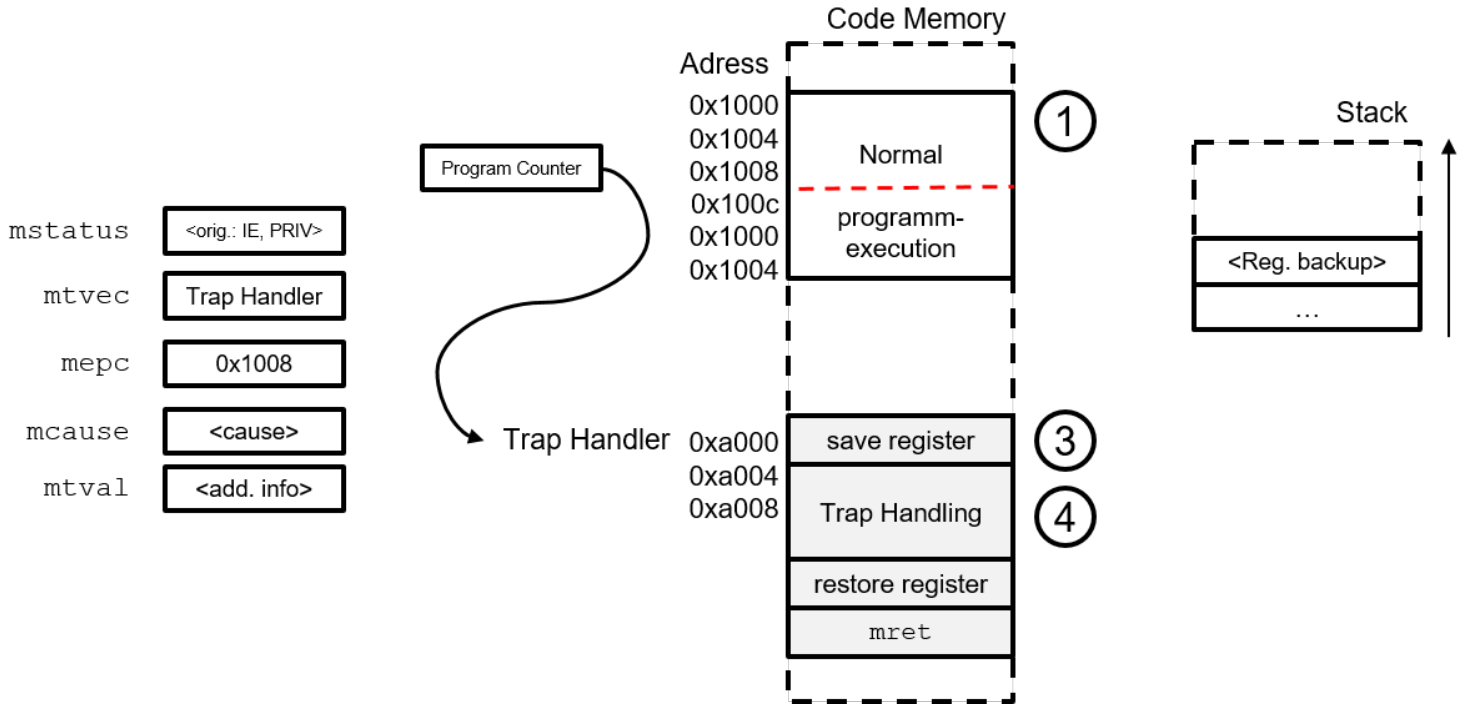
Trap Verarbeitung RISC-V

1. Die CPU arbeitet ein Programm ab
2. Interrupt während der Programmabar-beitung
3. ISR (Interrupt Service Routine) rettet Register



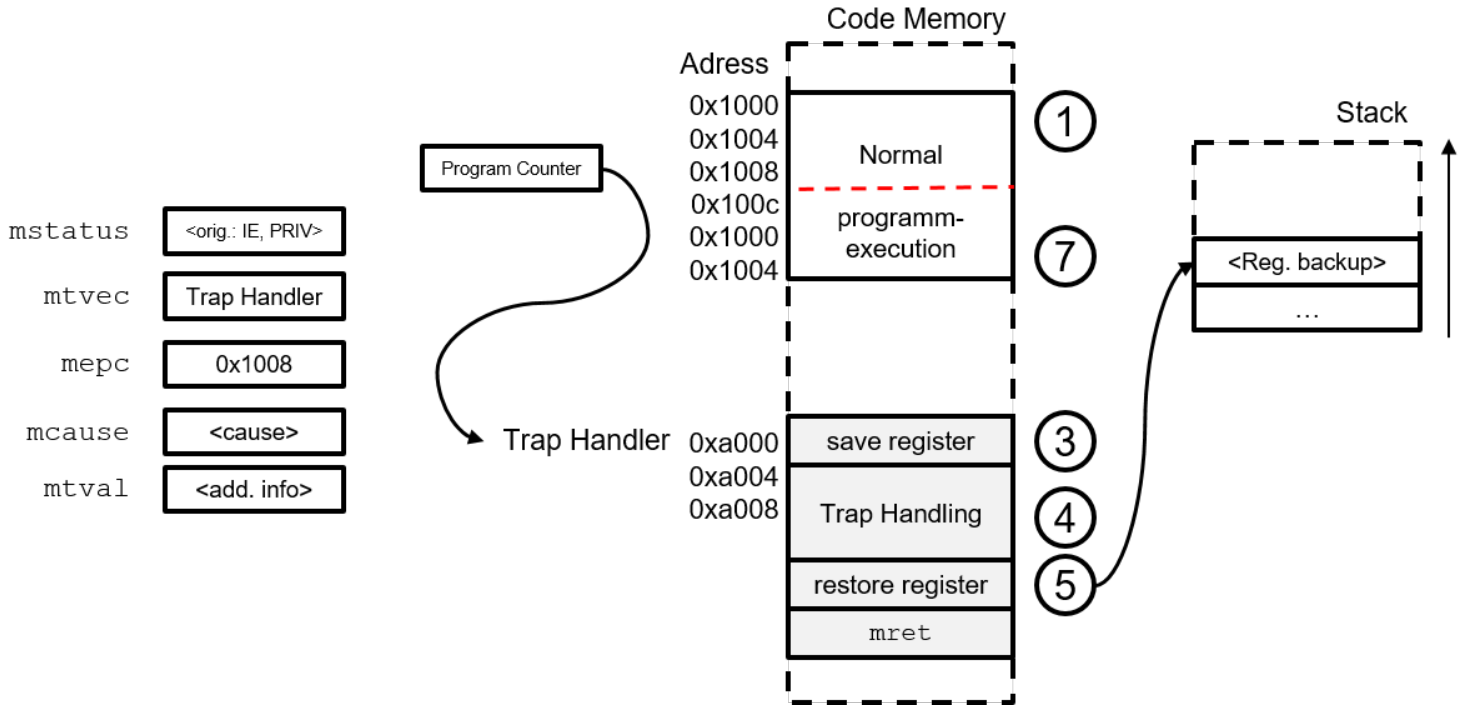
Trap Verarbeitung RISC-V

1. Die CPU arbeitet ein Programm ab
2. Interrupt während der Programmabar-beitung
3. ISR (Interrupt Service Routine) rettet Register
4. Eigentliche Interrupt-Bearbeitung



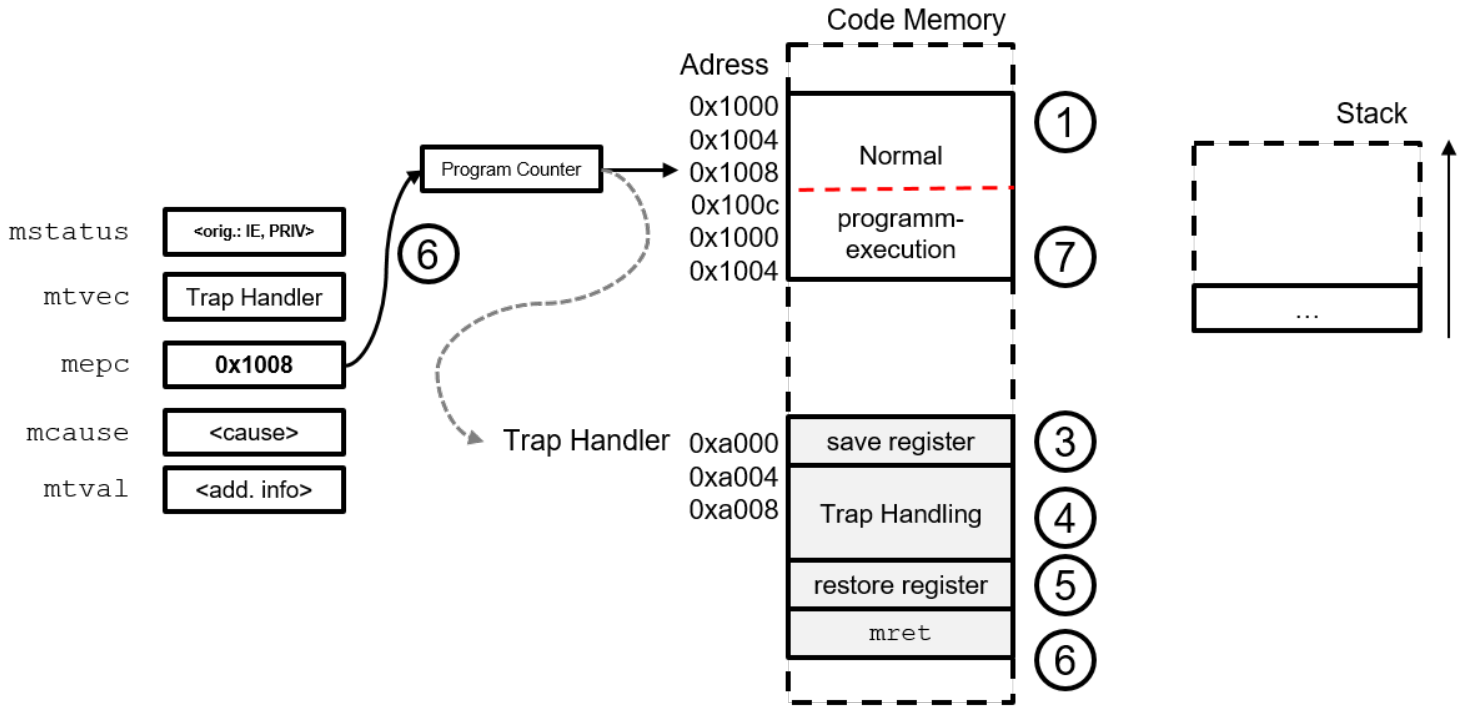
Trap Verarbeitung RISC-V

1. Die CPU arbeitet ein Programm ab
2. Interrupt während der Programmabar-beitung
3. ISR (Interrupt Service Routine) rettet Register
4. Eigentliche Interrupt-Bearbeitung
5. Gerettete Register restaurieren



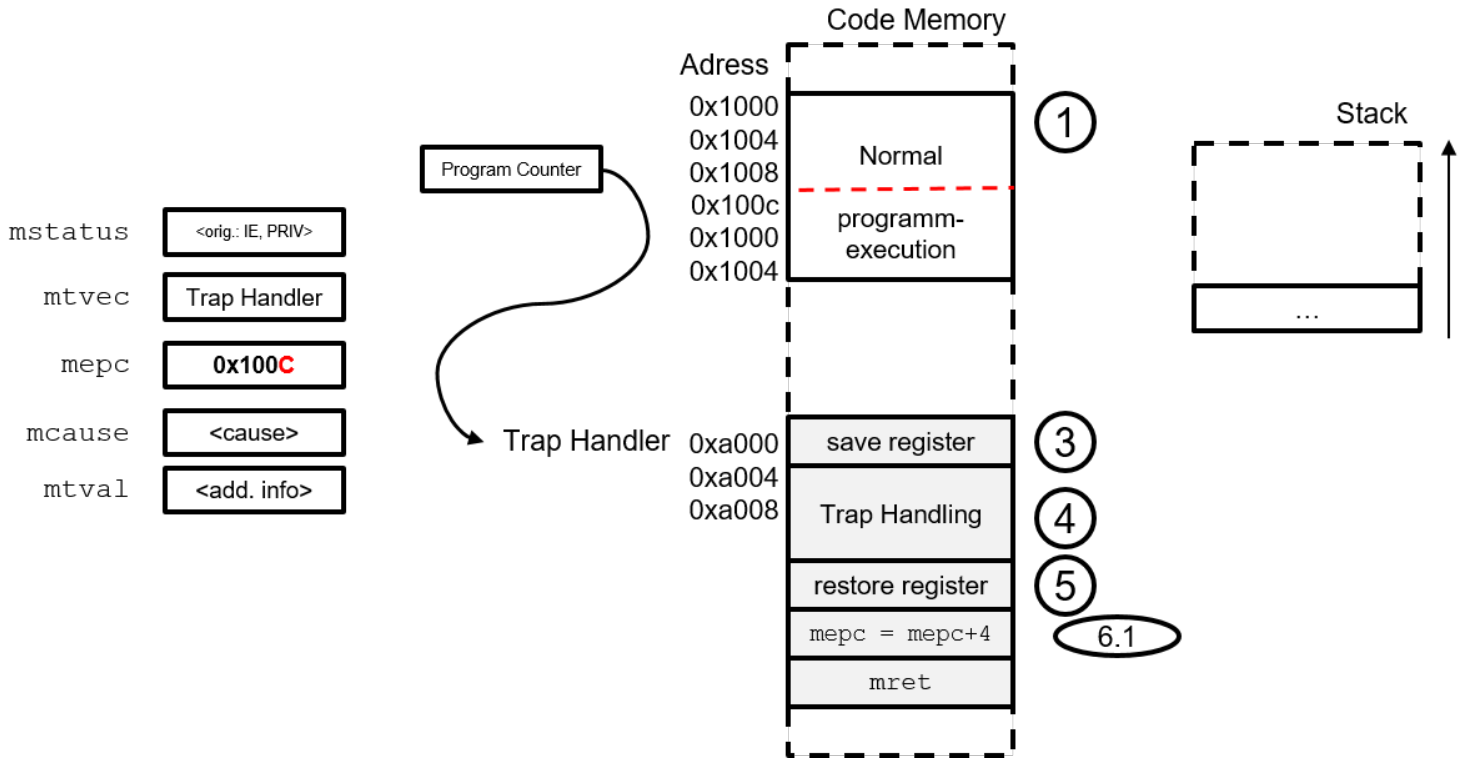
Trap Verarbeitung RISC-V

1. Die CPU arbeitet ein Programm ab
2. Interrupt während der Programmabar-beitung
3. ISR (Interrupt Service Routine) rettet Register
4. Eigentliche Interrupt-Bearbeitung
5. Gerettete Register restaurieren
6. Befehl *Return from Interrupt*



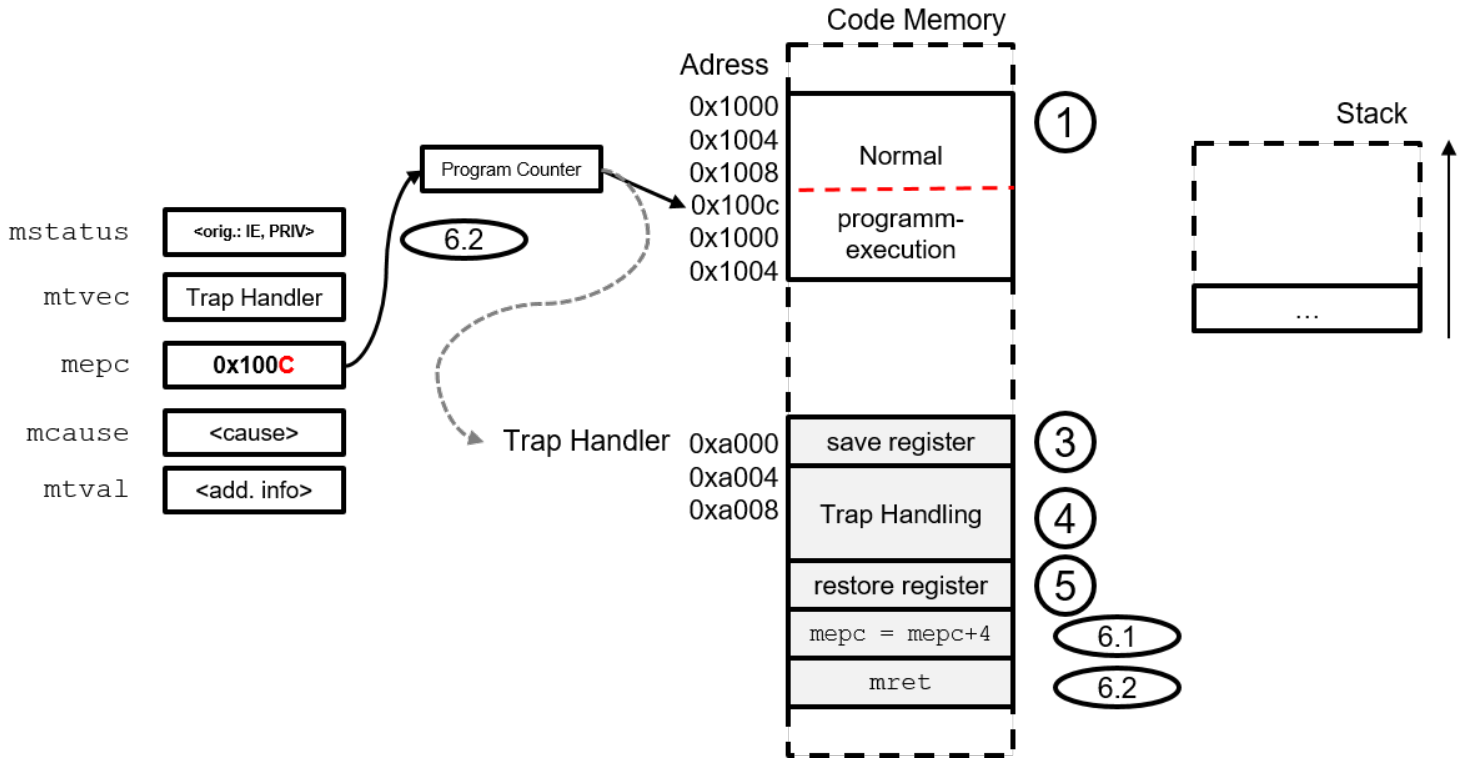
Trap Verarbeitung RISC-V [ODER]

1. Die CPU arbeitet ein Programm ab
2. Interrupt während der Programmabar-beitung
3. ISR (Interrupt Service Routine) rettet Register
4. Eigentliche Interrupt-Bearbeitung
5. Gerettete Register restaurieren
6. Befehl *Return from Interrupt*



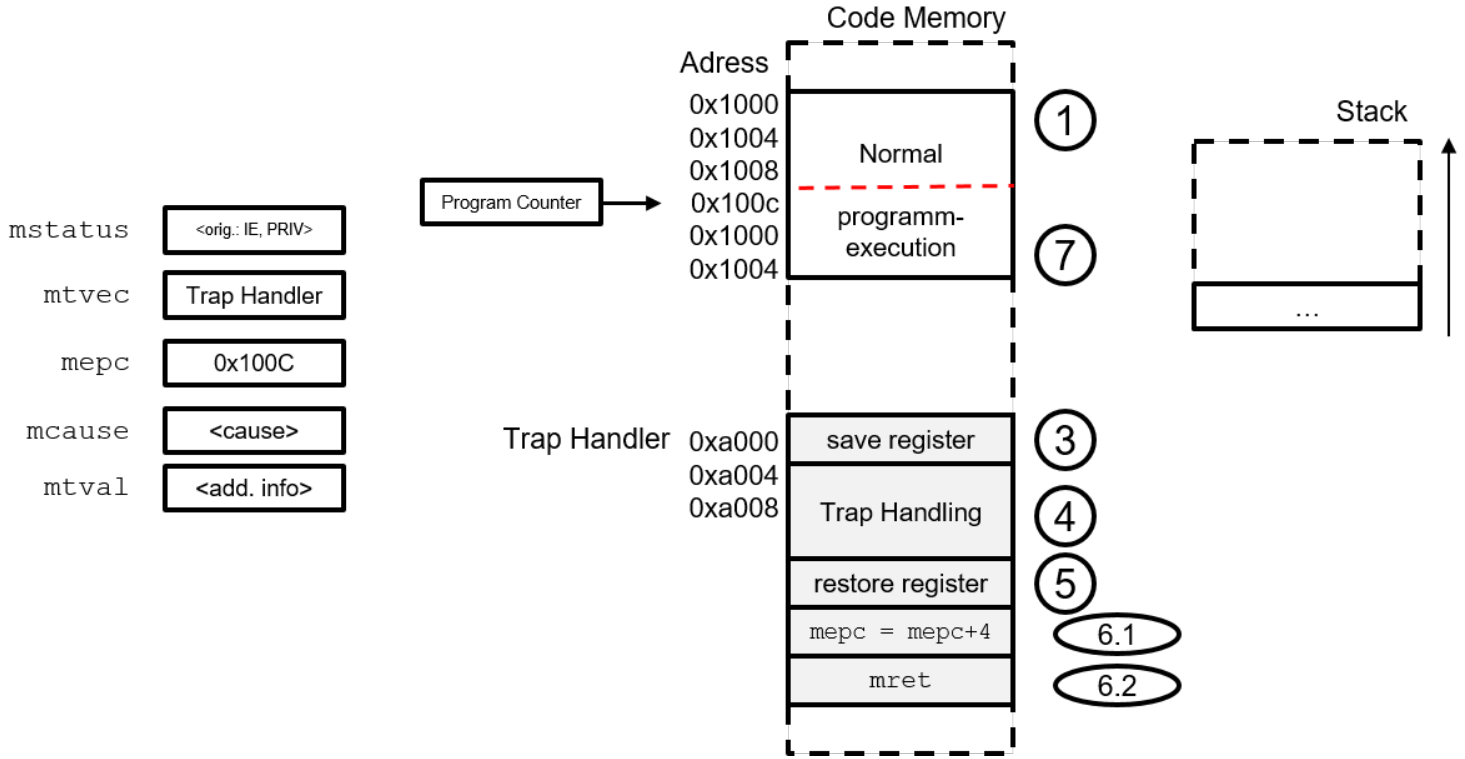
Trap Verarbeitung RISC-V [ODER]

1. Die CPU arbeitet ein Programm ab
2. Interrupt während der Programmabar-beitung
3. ISR (Interrupt Service Routine) rettet Register
4. Eigentliche Interrupt-Bearbeitung
5. Gerettete Register restaurieren
6. Befehl *Return from Interrupt*



Trap Verarbeitung RISC-V

2. Interrupt während der Programmabar-beitung
3. ISR (Interrupt Service Routine) rettet Register
4. Eigentliche Interrupt-Bearbeitung
5. Gerettete Register restaurieren
6. Befehl *Return from Interrupt*
7. Normalen Programmablauf fortsetzen



DEMO: BasicTrap (ecall)

- Am Beispiel eines einfachen `ecall` Aufrufs wird das Verhalten der RISC-V CPU bei der Verarbeitung einer synchronen Unterbrechung (Exception) gezeigt und erklärt
- Zusatz Informationen
  - Mittels `monitor` Befehl können im Debugger Infos aus dem Simulator abgefragt werden
    - `monitor cpu MSTATUS` -> interrupts are enabled globally
    - `monitor cpu MCAUSE` -> trap cause
    - `monitor cpu MTVEC` -> ...
  - Ausgabe auf die Console wird über ein Serielles Interface simuliert (UART)

Zusammenfassung CSR Befehle

- Infos aus: *The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA*
  - Chapter 9: Control and Status Register (CSR) Instructions, Version 2.0
  - Chapter 25: RISC-V Assembly Programmer's, Handbook, Table 25.3
- Alle CSR Instruktionen können ein CSR **atomar** Lesen-Verändern-Schreiben
  - `csrrw` ... Atomic Read/Write CSR (e.g. for atomically swap values)
    ```
    csrrw rd,csr,rs1
    ```
  - `csrrs` ... Atomic Read and Set Bits in CSR
    ```
    csrrs  rd,csr,rs1
    ```
  - `csrrc` ... Atomic Read and Clear Bits in CSR
    ```
    csrrc  rd,csr,rs1
    ```

CSR Pseudo-Befehle

- CSR Pseudoinstructions
  - `csrr` ... Read CSR
  - `csrw` ... Write CSR

WFI *Wait for Interrupt*

- Hinweis von der SW an die HW, dass es aktuell keine "sinnvollen" Aktivitäten auszuführen gibt und somit auf ein Interrupt gewartet werden kann
- Z.B. kann die HW in einen Energiesparmodus geschaltet werden.
- Die eigentliche Umsetzung ist RISC-V Prozessor spezifisch, d.h.
  - stoppen der Clock bis zu einem Interrupt bzw. ein Interrupt Pending wird
  - einfache Implementierungen machen daraus einen `nop`
  - => daher ist `wfi` oft in einer Loop eingebaut die `mip` prüft

WFI *Wait for Interrupt*

- Mittels eines Interrupt wird der "Sparmodus" wieder verlassen
- Abhängig von `status.MIE` (globales Interrupt-Enable Bit) zum Zeitpunkt der Ausführung von `wfi` verhält sich der Prozessor wie folgt
  - `status.MIE` == 1 ... Interrupts sind enabled, d.h. es wird bei einem Interrupt der entsprechende Trap angesprungen
  - `status.MIE` == 0 ... Interrupts sind disabled, d.h. es wird bei einem Interrupt (Pending), direkt nach `wfi` weitergemacht. Hier wird meist dann `mip` ausgewertet und die entsprechende Reaktion eingeleitet.

Was ist notwendig damit einen Interrupt zum Trap führt?

- Damit ein Interrupt (asynchrone Unterbrechung) im M-Mode einen Trap auslöst, müssen die entsprechenden Enable Bits gesetzt sein
  - `mstatus.MIE` == 1 ... globales Interrupt-Enable Bit
  - `mie.MTIE` == 1 ... **Timer** Interrupt-Enable Bit
  - `mie.MSIE` == 1 ... **Software** Interrupt-Enable Bit
  - `mie.MEIE` == 1 ... **External** Interrupt-Enable Bit
- Ein entsprechender Interrupt Signalisiert werden (Interrupt ReQuest, IRQ)
  - `mie.MTIP` == 1 ... **Timer** Interrupt-Pending Bit
  - `mie.MSIP` == 1 ... **Software** Interrupt-Pending Bit
  - `mie.MEIP` == 1 ... **External** Interrupt-Pending Bit

Danke für Ihre Aufmerksamkeit!
Fragen?