计算机组成与设计

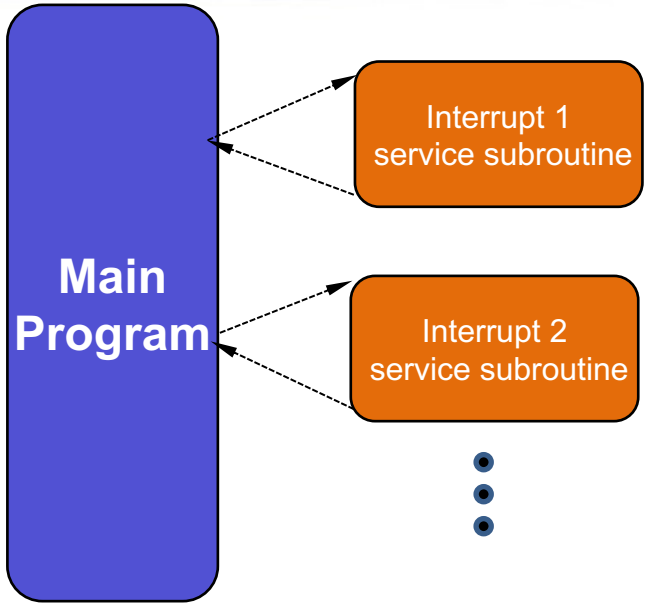
Computer Organization & Design

The Hardware/Software Interface

Chapter 4
Interrupt and Exception

马德
College of Computer Science and Technology
Zhejiang University
made@zju.edu.cn

Processor Design…



CPU within Exception

4.9 Exception (Interruption)

- **The cause of changing CPU's work flow :**
  - Control instructions in program (bne/beq, j/ jal , etc)

    It is foreseeable in programming flow
  - Something happen suddenly (Exception and Interruption)

    It is unpredictable
    - Call Instructions triggered by hardware
- **Unexpected events**
  - Exception: from within processor (overflow, undefined instruction, etc)
  - Interruption : from outside processor (input /output)

Exception

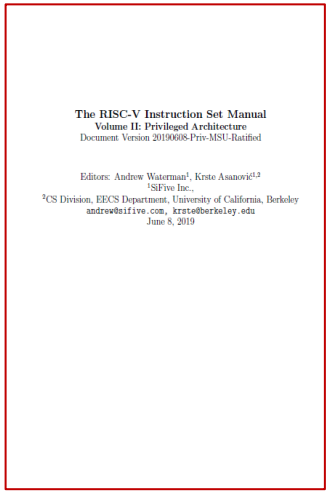
- **Exception**

  An Exception is a unexpected event from within processor. (狭义的异常)

- **We follow the RISC convention**
  - using the term exception to refer to any unexpected change in control flow (广义的异常，包含interupt)
- **Here we will discuss two types exceptions :**
  - undefined instruction、arithmetic overflow
  - Interruption by IO、System call
- **Interruption**
  - Implementation methods are similar

How Exceptions Are Handled

- **What must do the processor?**
  - When exception happens the processor must do something
  - The predefined process routines are saved in memory when computer starts
- **Problem**
  - how can CPU go to relative routine when an exception occurs
  - CPU should know
    - The cause of exception
    - which instruction generate the exception
  - Like the “jal" instruction, but the hardware triggers


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 Privileged

All hardware implementations must provide M-mode

- **RISC-V Privileged Architecture**
  - The machine level has the highest privileges
    - and is the only mandatory privilege level for a RISC-V hardware platform.
    - Code run in machine-mode (M-mode) is usually inherently trusted, as it has low-level access to the machine implementation.
    - M-mode can be used to manage secure execution environments on RISC-V.
  - User-mode (U-mode) and supervisor-mode (S-mode) are intended for conventional application and operating system usage respectively.

Level	Encoding	Name	Abbreviation	
0	00	User/Application	U	用户模式
1	01	Supervisor	S	监督模式
2	10	Reserved		保留
3	11	Machine	M	机器模式

RISC-V Privilege Modes Usage

- **Each privilege level has**
  - a core set of privileged ISA extensions
    - with optional extensions and variants.
- **Combinations of privilege modes**
  - Implementations might provide anywhere from 1 to 3 privilege modes
    - trading off reduced isolation for lower implementation cost

Number of levels	Supported Modes	Intended Usage
1	M	Simple embedded systems
2	M, U	Secure embedded systems
3	M, S, U	Systems running Unix-like operating systems

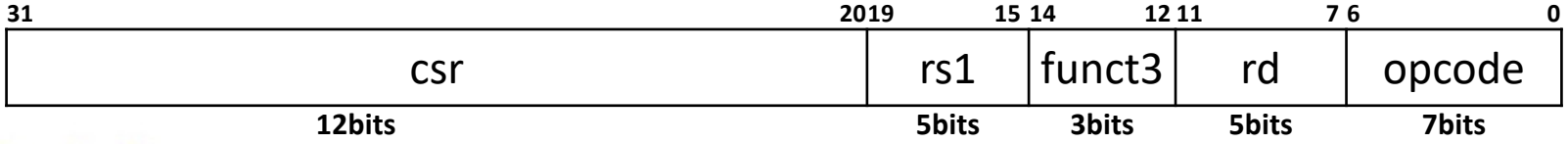
  - For example, machine-mode supports an optional standard extension for memory protection.

RISC-V interrupt structure

- **All hardware implementations must provide M-mode**
  - as this is the only mode that has unfettered access to the whole machine.
  - The simplest RISC-V implementations may provide only M-mode
    - though this will provide no protection against incorrect or malicious application code
- **Machine mode(M-mode) most important task**
  - that is to intercept and handle interrupts/exceptions
  - There are 4096 Control and Status Registers (CSRs)
    - Similar to CP0 of MIPS

Control and Status Registers (CSRs)

- **CSRs, are additional set of registers**
  - accessible by some subset of the privilege levels using the CSR instructions
  - These can be divided into two main classes:
    - those that atomically read-modify-write control and status registers
    - and all other privileged instructions.
- **All privileged instructions encode by the SYSTEM major opcode**
  - CSRs and instructions are associated with one privilege level, they are also accessible at all higher privilege levels
    - The standard RISC-V ISA sets aside a 12-bit encoding space (csr[11:0]) for up to 4,096 CSRs.



CSR Instruction

- **All CSR instructions atomically read-modify-write a single CSR**
  - whose CSR specifier is encoded in the 12-bit csr field of the instruction held in bits 31–20
    - The immediate forms use a 5-bit zero-extended immediate encoded in the rs1 field.

	31 20	19 15	14 12	11 7	6 0
	csr	rs1	funct3	rd	opcode
	12bits	5bits	3bits	5bits	7bits
I: CSRRW	csr	rs1	001	rd	1110011
I: CSRRS	csr	rs1	010	rd	1110011
I: CSRRC	csr	rs1	011	rd	1110011
I: CSRRWI	csr	zimm	101	rd	1110011
I: CSRRSI	csr	zimm	110	rd	1110011
I: CSRRCI	csr	zimm	111	rd	1110011

Interrupts Instruction

- **Exception return**
  - MRET
    - PC <= MEPC；(MStatus寄存器有变化)

	31 25	24 20	19 15	14 12	11 7	6 0
R：MRET	0011000	00010	00000	000	00000	1110011
R：SRET	0001000	00010	00000	000	00000	1110011
R：wfi	0001000	00101	00000	000	00000	1110011

- **Environment call**
  - ecall
    - MEPC = ecall指令本身的PC值
- **Breakpoint**
  - ebreak
    - MEPC = ebreak指令本身的PC值

	31 25	24 20	19 15	14 12	11 7	6 0
I：ecall	0000000	00000	00000	000	00000	1110011
I：ebreak	0000000	00001	00000	000	00000	1110011

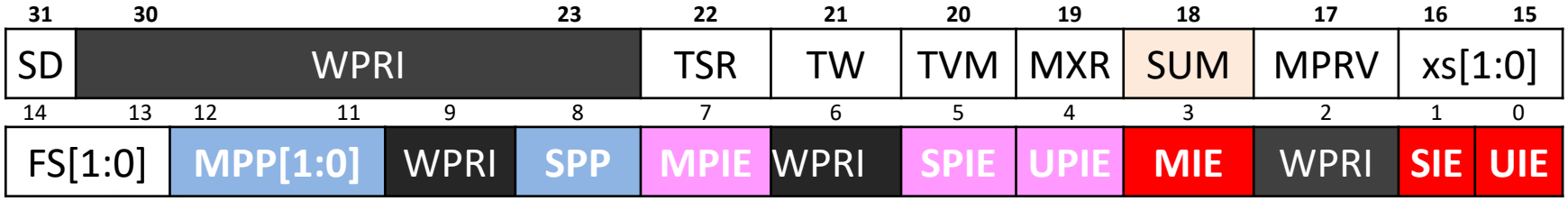
Exception & Interrupt related registers

CSR	Privilege	Abbr.	Name	Description
0x300	MRW	mstatus	Machine STATUS register 机器模式状态寄存器	MIE、MPIE域标记中断全局使能
0x304	MRW	mie	Machine Interrupt Enable register 机器模式中断使能寄存器	控制不同类型中断的局部使能
0x305	MRW	mtvec	Machine trap-handler base address 机器模式异常入口基地址寄存器	进入异常服务程序基地址
0x341	MRW	mepc	Machine exception program counter 机器模式异常PC寄存器	异常断点PC地址
0x342	MRW	mcause	Machine trap cause register 机器模式原因寄存器	处理器异常原因
0x343	MRW	mtval	Machine Trap Value register 机器模式异常值寄存器	处理器异常值地址或指令
0x344	MRW	mip	Machine interrupt pending 机器模式中断挂起寄存器	处理器中断等待处理

Interrupt Registers： mstatus(*0x300*)

- Machine STATUS register
  - These bits are primarily used to guarantee atomicity with respect to interrupt handlers in the current privilege mode.

bit	Name	Attributes	Description
0	UIE	RW	User interrupt enable
1	SIE	RW	Supervisor interrupt enable
3	MIE	RW	Machine interrupt enable
4	UPIE	RW	previous user-level interrupts enable
5	SPIE	RW	Previous Supervisor interrupt-enable
7	MPIE	RW	Previous Machine interrupt enable
8	SPP	RW	Supervisor Previous Privilege mode
12:11	MPP	RW	Machine Previous Privilege mode
……			
31:23,9,6,2	WPRI		Reserved



Interrupt Registers： mie/mip(*0x304/344*)

- **Machine Interrupt Enable register**
  - MIE register controls whether it can respond to interrupts, corresponding different modes
    - MEIE、SEIE and UEIE enable external interrupt
    - MSIE、SSIE & USIE enable software interrupts
    - MTIE、STIE and UTIE enable timer interrupts
- **Machine interrupt-pending register**
  - The mip register is an MXLEN-bit read/write register containing information on pending interrupts

	31 … 12	11	10	9	8	7	6	5	4	3	2	1	0
mie	WPRI	**MEIE**	WPRI	**SEIE**	UEIE	**MTIE**	WPRI	**STIE**	UTIE	**MSIE**	WPRI	**SSIE**	USIE
mip	WPRI	**MEIP**	WPRI	**SEIP**	UEIP	**MTIP**	WPRI	**STIP**	UTIP	**MSIP**	WPRI	**SSIP**	USIP

M, S and U correspond to M mode, S mode and U mode interrupt respectively

Interrupt Registers： mtvec (*0x305*)

Machine Trap-Vector Base-Address Register

- The mtvec register holds trap vector configuration, consisting of a vector base address (BASE) and a vector mode (MODE)

- The value in the BASE field must always be aligned on a 4-byte boundary, and the MODE setting may impose additional alignment constraints on the value in the BASE field.

MODE Value	Name	Description
0	Direct(查询)	All exceptions set pc to BASE
1	Vectored(向量)	Asynchronous interrupts set pc to BASE+4×cause
≥2	--	Reserved

BASE+ 4×Exception

Interrupt Registers： mepc (*0x341*)

- **Machine Exception Program Counter**
  - mepc is a WARL register that must be able to hold all valid physical and virtual addresses.
    - When a trap is taken into M-mode, mepc is written with the address of the instruction that was interrupted or exception.

  - The low bit of mepc (mepc[0]) is always zero.
    - On implementations that support only IALIGN=32, the two low bits (mepc[1:0]) are always zero.
  - Exception： `mepc ← pc` （狭义的异常）
  - Interrupted： `mepc ← pc + 4`

Interrupt Registers： mcause (*0x342*)

- **Machine Cause Register (mcause)**
  - When a trap is taken mcause register is written with a code indicating the event that caused the Exception/Interruption.

  - Exception Code与mtvec的向量模式相对应
    - 在异步中断时，不同的模式会跳转到不同的入口
  - The Exception Code is a WLRL
    - Write/Read Only Legal Values
    - 如果写入值不合法可以引发非法指令异常

Exception Code mcause (*0x342*)

INT	E 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	16-23	Reserved for future standard use
0	32-47	Reserved for future standard use
0	14/≥64	Reserved for future standard use

INT	E Code	Description
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	15	Store/AMO page fault
0	24-31	Reserved for custom use
0	48-63	Reserved for custom use

RISV-V Interrupt priority

External interrupt > Software interrupt > Timer interrupt

Synchronous exception priority

Priority	Exception Code	Description
Highest	3	Instruction address breakpoint
	12	Instruction page fault
	1	Instruction access fault
	2	Illegal instruction
	0	Instruction address misaligned
	8，9，11	Environment call
	3	Environment break
	3	Load/Store/AMO address breakpoint
	6	Store/AMO address misaligned
	4	Load address misaligned
	15	Store/AMO page fault
	13	Load page fault
	7	Store/AMO access fault
Lowest	5	Load access fault

RISC-V中断处理--进入异常

- **RISC-V处理器检测到异常，开始进行异常处理：**
  - 停止执行当前的程序流，转而从CSR寄存器mtvec定义的PC地址开始执行；
  - 更新机器模式异常原因寄存器：mcause
  - 更新机器模式中断使能寄存器：mie
  - 更新机器模式异常PC寄存器：mepc
  - 更新机器模式状态寄存器：mstatus
  - 更新机器模式异常值寄存器：mtval

(以上更新均有硬件完成)

RISC-V中断结构--退出异常

- 异常程序处理完成后，需要从异常服务程序中退出，并返回主程序
  - RISCV中定义了一组退出指令MRET，SRET，和URET
  - 机器模式对应MRET。
- 机器模式下退出异常(MRET)
  - 程序流转而从csr寄存器mepc定义的pc地址开始执行
  - 同时硬件更新csr寄存器机器模式状态寄存器mstatus
    - 寄存器MIE域被更新为当前MPIE的值：mie ← mpie
    - MPIE 域的值则更新为1： MPIE ← 1

4.9 Exceptions and Interrupts

- **"Unexpected" events requiring change in flow of control**
  - Different ISAs use the terms differently
- **Exception**
  - Arises within the CPU
    - e.g., undefined opcode, syscall, …
- **Interrupt**
  - From an external I/O controller
- **Dealing with them without sacrificing performance is hard**

Handling Exceptions

- **Save PC of offending (or interrupted) instruction**
  - In RISC-V: Supervisor Exception Program Counter (SEPC)
- **Save indication of the problem**
  - In RISC-V: Supervisor Exception Cause Register (SCAUSE)
  - 64 bits, but most bits unused
    - Exception code field: 2 for undefined opcode, 12 for hardware malfunction, …
- **Jump to handler**
  - Assume at 0000 0000 1C09 0000_{hex}

An Alternate Mechanism

- **Vectored Interrupts**
  - Handler address determined by the cause
- **Exception vector address to be added to a vector table base register:**
  - Undefined opcode 00 0100 0000_{two}
  - Hardware malfunction: 01 1000 0000_{two}
  - …: …
- **Instructions either**
  - Deal with the interrupt, or
  - Jump to real handler

典型处理器中断结构：向量模式

- **Intel x86中断结构**
  - 间接向量：000～3FF，占内存最底1KB空间
    - 每个向量由二个16位生成20位中断地址
    - 共256个中断向量，向量编号n=0～255
    - 分硬中断和软中断，响应过程类同，触发方式不同
    - 硬中断响应由控制芯片8259产生中断号n(接口原理课深入学习)
- **ARM中断结构**
  - 固定向量方式(嵌入式课程深入学习)

异常类型	偏移地址(低)	偏移地址(高)	
复 位	00000000	FFFF0000	
未定义指令	00000004	FFFF0004	
软中断	00000008	FFFF0008	
预取指令终	0000000C	FFFF000C	
数据终止	00000010	FFFF0010	
保留	00000014	FFFF0014	
中断请求(IRQ)	00000018	FFFF0018	
快速中断请求(FIQ)	0000001C	FFFF001C	

Simplify Interrupt Design

- 简化中断设计
  - 采用ARM中断向量(不兼容RISC-V)
    - 实现非法指令异常和外中断
    - 设计EPC
  - 兼容RISC-V*
    - 仅M-Mode中断寄存器(MCause、Mstatus、MIE、MIP、MEPC和MTVEC)
    - 设计mret、CSRRW (csrw rd, csr, rs1)和ecall指令
- ARM中断向量表

向量地址	ARM异常名称	ARM系统工作模式	本课程定义
0x0000000	复位	超级用户Svc	复位(M-MODE)
0x0000004	未定义指令终止	未定义指令终止Und	非法指令异常
0x0000008	软中断（SWI）	超级用户Svc	ECALL
0x000000c	Prefetch abort	指令预取终止Abt	Int外部中断（硬件）
0x0000010	Data abort	数据访问终止Abt	Reserved自定义
0x0000014	Reserved	Reserved	Reserved自定义
0x0000018	IRQ	外部中断模式IRQ	Reserved自定义
0x000001C	FIQ	快速中断模式FIQ	Reserved自定义

Handler Actions

- **Read cause, and transfer to relevant handler**
- **Determine action required**
- **If restartable**
  - Take corrective action
  - use SEPC to return to program
- **Otherwise**
  - Terminate program
  - Report error using SEPC, SCAUSE, …

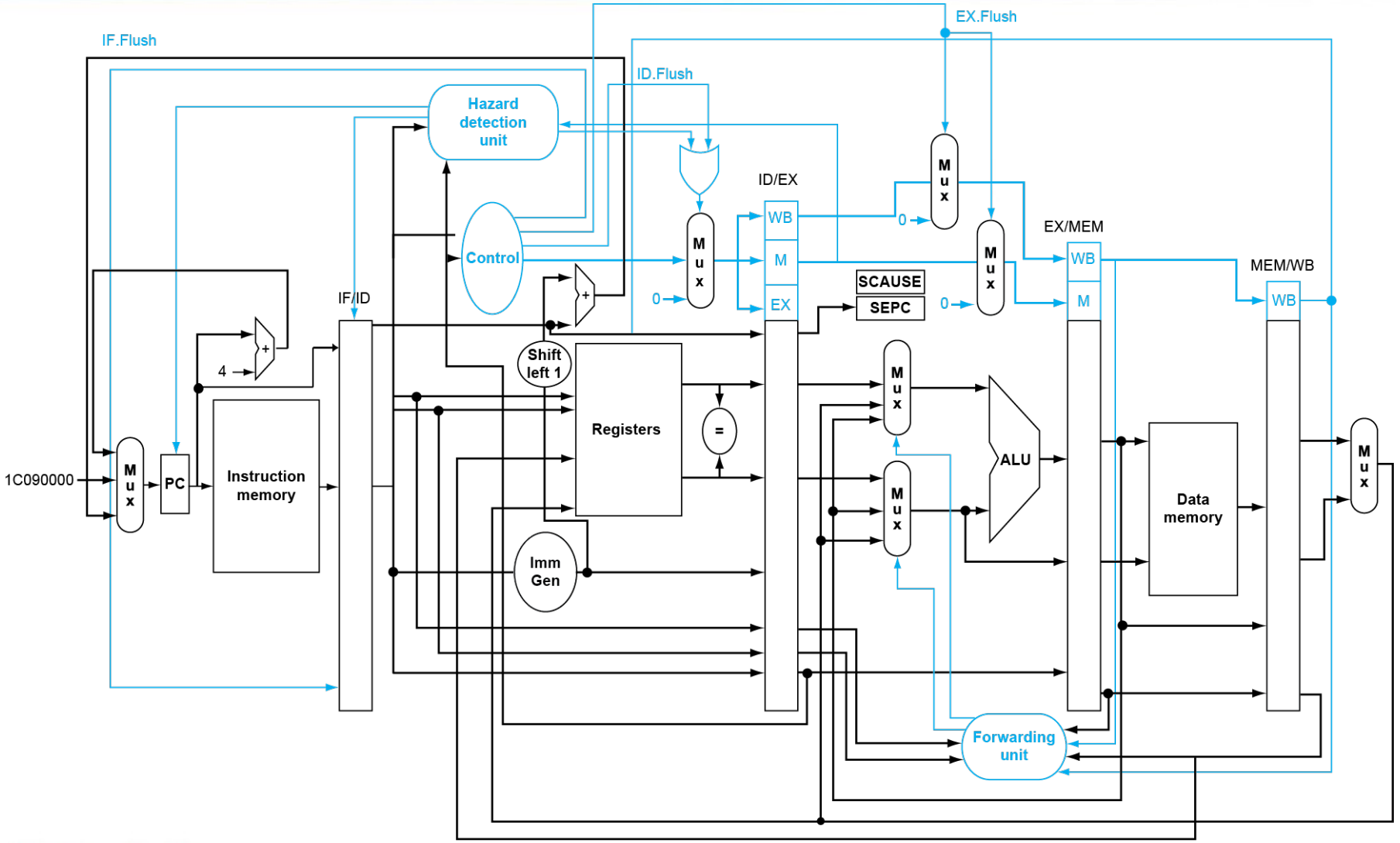
Exceptions in a Pipeline

- Another form of control hazard
- Consider malfunction on add in EX stage

```
add x1, x2, x1
```

  - Prevent x1 from being clobbered
  - Complete previous instructions
  - Flush `add` and subsequent instructions
  - Set SEPC and SCAUSE register values
  - Transfer control to handler
- Similar to mispredicted branch
  - Use much of the same hardware

Pipeline with Exceptions



Exception Properties

- **Restartable exceptions**
  - Pipeline can flush the instruction
  - Handler executes, then returns to the instruction
    - Refetched and executed from scratch
- **PC saved in SEPC register**
  - Identifies causing instruction

Exception Example

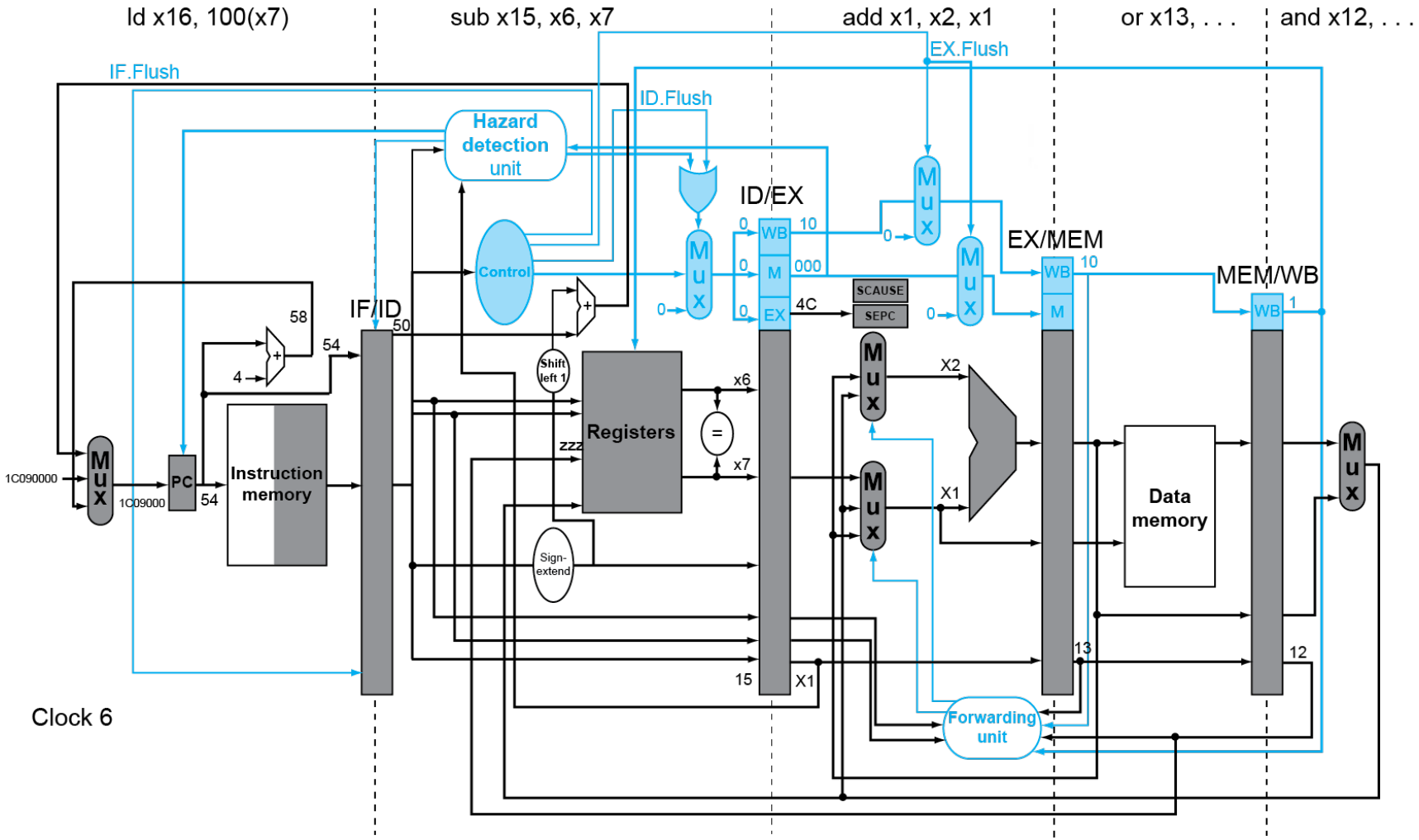
- Exception on **add** in

```
40      sub   x11, x2, x4
44      and   x12, x2, x5
48      orr   x13, x2, x6
4c      add   x1,  x2, x1
50      sub   x15, x6, x7
54      ld    x16, 100(x7)
…
```

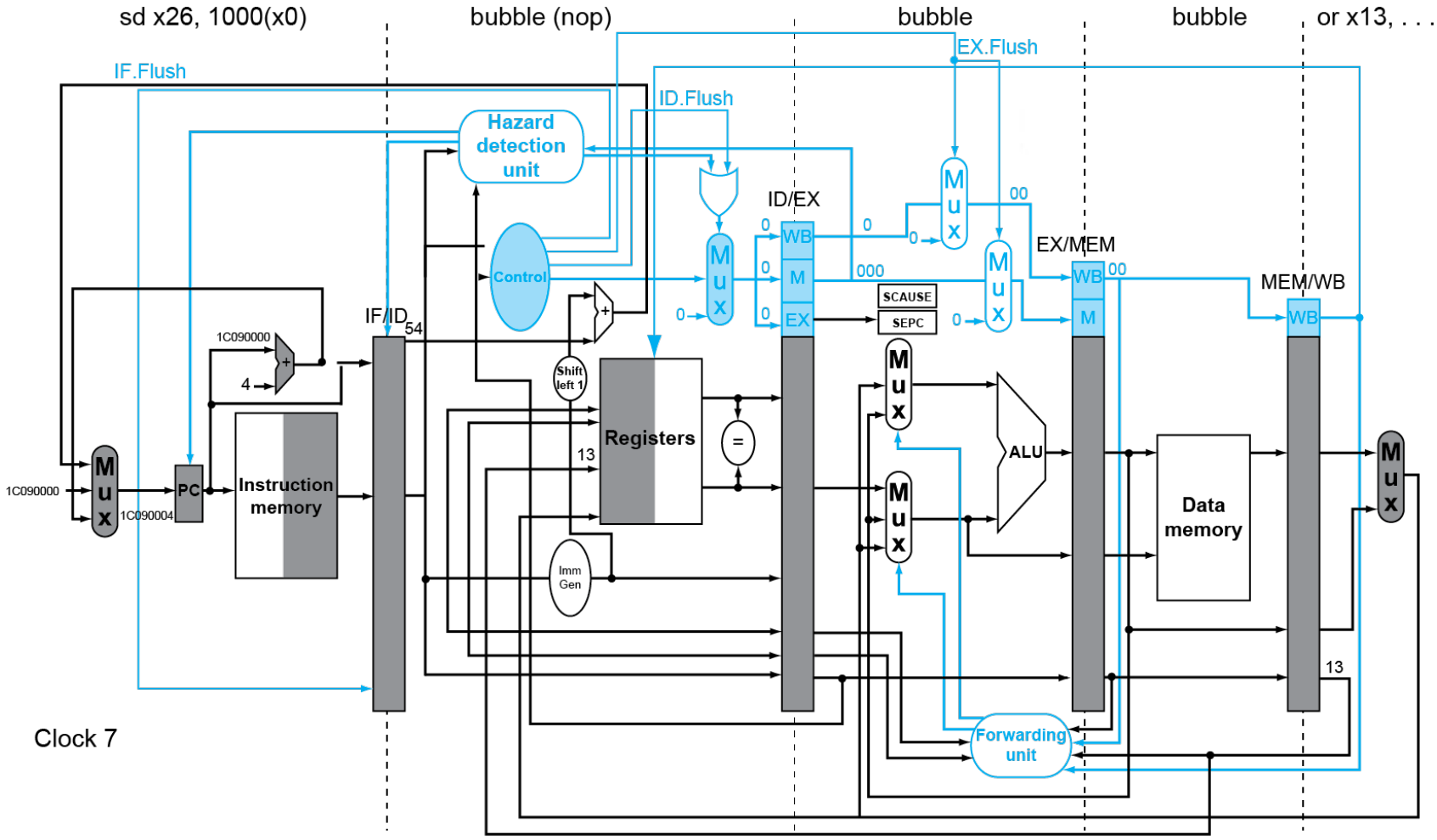
- Handler

```
1c090000    sd  x26, 1000(x10)
1c090004    sd  x27, 1008(x10)
…
```

Exception Example



Exception Example



Multiple Exceptions

- **Pipelining overlaps multiple instructions**
  - Could have multiple exceptions at once
- **Simple approach: deal with exception from earliest instruction**
  - Flush subsequent instructions
  - "Precise" exceptions
- **In complex pipelines**
  - Multiple instructions issued per cycle
  - Out-of-order completion
  - Maintaining precise exceptions is difficult!

Imprecise Exceptions

- **Just stop pipeline and save state**
  - Including exception cause(s)
- **Let the handler work out**
  - Which instruction(s) had exceptions
  - Which to complete or flush
    - May require “manual” completion
- **Simplifies hardware, but more complex handler software**
- **Not feasible for complex multiple-issue out-of-order pipelines**


END