Functions:

- $sa(c) = I(c)[10:6]$ shift amount
- $f(c) = I(c)[5:0]$ function code
- $d(c) =$ access width

3 Interrupts

Notes:

- Interrupts 'kind of' procedure call [MP00]

Def:

- Change of control flow due to event signals $ev[j]$

Number interrupts using indices $j \in \{0, \ldots, 31\}$

j	name	*symbol*	external	maskable	resume-type
0	reset	$reset$	yes	no	abort
1	illegal instruciton	ill	no	no	abort
2	misalignment	$misa$	no	no	abort
3	page fault fetch	pff	no	no	repeat
4	page fault l/s	$pfls$	no	no	repeat
5	trap	$trap$	no	no	continue
6	arith. overflow	ovf	no	yes	continue/abort
$6+i$	ext. I/O	ex_i	yes	yes	continue

Priority of interrupts:

- priority of interrupt specified by index j
- goal: $j < j' \Rightarrow j$ receives service before j'

Classify the set of indices:

- maskable / not maskable
  denote the set of indices of maskable interrupts by M
- external / internal
  denote the set of indices of external interrupts by E

External event signals $eev[j]$ where $j \in E$
Internal event signals $iev[j]$ where $j \notin E$

Generation of interrupts:
How are interrupts generate?

- Activation of $iev[j]$ is determined by the configuration c only: $iev(c)[j]$
- $eev[j]$ generated by I/O device / external source
  (assume $eev[j]^t$ well defined)

Internal interrupts $iev(c)[j]$:

$$
\begin{array}{lll}
iev(c)[1] = & ill(c) & = \text{Exercise} \\
iev(c)[2] = & misa(c) & = c.dpc[0] \vee c.dpc[1] \vee (load(c) \vee store(c) \wedge (d(c) \upharpoonright ea(c))) \\
iev(c)[3] = & pff(c) & = \text{done by MMU (later)} \\
iev(c)[4] = & pfls(c) & = \text{done by MMU (later)} \\
iev(c)[5] = & trap(c) & = (opc(c) = 111110) \\
iev(c)[6] = & ovf(c) & = (\text{see [MP00]})
\end{array}
$$

Handling interrupts:

1. instruction i is interrupted
2. save current state
3. jump to interrupt service routine (ISR) which handles interrupt
4. restore state: depending on resume-type
   - abort: abort execution
   - repeat: execute i again
   - continue: execute $i + 1$

3.1 New Configuration

Extend configuration c by new interrupt specific registers:

$$c = (c.pc, c.dpc, c.gpr, c.m, c.spr)$$

a	$spr(a)$	name
0	SR	status register
1	ESR	exception status register
2	ECA	exception cause register
3	EPC	exception pc
4	$EDPC$	exception dpc
5	$Edata$	exception data register
6	PTL	page table length
7	PTO	page table origin
8	$EMODE$	exception mode register
9	$MODE$	mode register

3.2 New Transition Function

New ISA transition function with interrupts:

$$c' = \delta_S(c, eev)$$

Moving data using 2 new instructions:

- gpr → spr: movi2s instruction
- spr → gpr: movs2i instruction

Formally:

$$movi2s(c) = (rtype(c) \wedge f(c) = 010001)$$
$$movi2s(c) \Rightarrow c'.spr(x) = \begin{cases} c.gpr(RS1(c)) & x = sa(c) \\ c.spr(x) & \text{otherwise} \end{cases}$$

$$movs2i(c) = (rtype(c) \wedge f(c) = 010000)$$
$$movs2i(c) \Rightarrow c'.gpr(x) = \begin{cases} c.spr(sa(c)) & x = RD(c) \\ c.gpr(x) & \text{otherwise} \end{cases}$$

3.2.1 Handling Interrupts

The cause vector ca is a function of c and eev:

$$ca(c, eev)[j] = \begin{cases} eev[j] & j \in E \\ iev(c)[j] & \text{otherwise} \end{cases}$$

The masked cause mca is computed from ca using the interrupt mask in the status register ($c.SR$).
If interrupt j is maskable and $c.SR[j] = 0$, then j is masked out:

$$mca(c, eev)[j] = \begin{cases} ca(c, eev)[j] \wedge c.SR[j] & j \in M \\ ca(c, eev)[j] & \text{otherwise} \end{cases}$$

If any mca bit is on, the jump to interrupt service routine (JISR) bit is set:

$$JISR(c, eev) = \bigvee_j mca(c, eev[j])$$

Interrupt lines might become active simultaneously

Need to know the smallest index of an active bit in mca.
Index is called the *interrupt level* il:

$$il(c, eev) = \min\{j \mid mca(c, eev)[j] = 1\}$$

il specifies the interrupt of highest priority which is to be serviced immediately
If $JISR(d, eev)$ holds:

- save dpc, pc, edata, mca
- switch to system mode
- mask all maskable interrupts
- execute ISR (handles $il(c, eev)$)

4 Excursion to OS Kernel

...