CS 51 Homework 8

Jessie Li

November 18, 2024

1A.

I added a new register to store a privilege bit `priv` and circuitry to ensure that `priv` is reset to 0 when the machine first starts to run and `priv` must be zero for any I/O operation.

Testing

`priv-io.ys` is a small test program derived from `privdemo-F24.ys` to demonstrate that I/O can only be accessed when `priv` = 0. It comes after the implementation of Q1B.

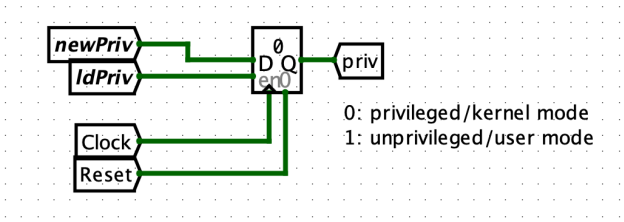

Figure 1: New privilege register

1B.

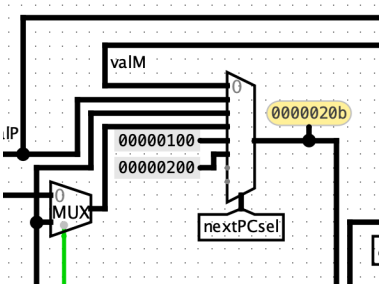

Figure 2: Special PC updates for system instructions.

I added support for two new instructions, `sysexit` and `sysexit`. `sysenter` sets the privilege mode to 1 and increments the PC by 1. `sysenter` clears the privilege mode to 0 and does the equivalent of `call 0x100` while saving PC+1 to the stack as the return address. Since both instructions have `icode` = `0xd`, I allow the FSM to differentiate between them by computing a new `icode` for each, `0xd` + `ifun`. `sysexit` is encoded as `0xd` and `sysenter` as `0xe`. I also added three new bits to the microinstructions:

- `ldPriv`: Asserted by instructions that change the privilege mode
- `newPriv`: The next value of privilege `priv` to be stored when the clock rises and `ldPriv` = 1
- `sysPCsel`: An extra bit that extends `newPCsel` to enable special system-related PC updates, such as calling `0x100` on `sysenter`

Testing

See `sys.ys` for a simple program that invokes `sysenter` and `sysexit`.

1C.

The new hardware line `exception` is asserted when an instruction uses an illegal register code:

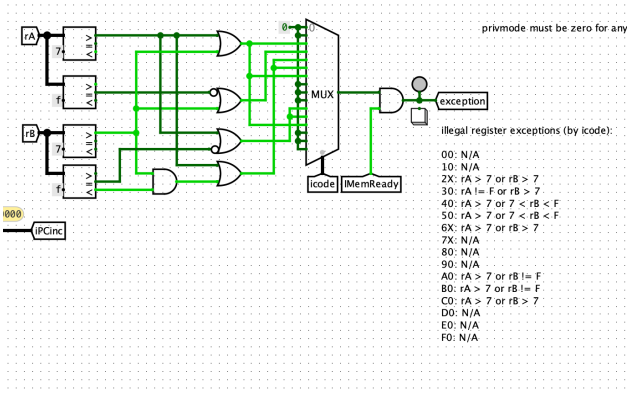

Figure 3: Exception checking

On the rising edge of `exception`, if `priv` is 1, the FSM enters the EXCEPTION state, clears `priv` to 0, and traps to special privileged code at address `0x200`without saving the current or would-be next PC, assuming no need to return (halts within the trap, since this is a fatal error). If `priv` is 0, the machine simply halts.

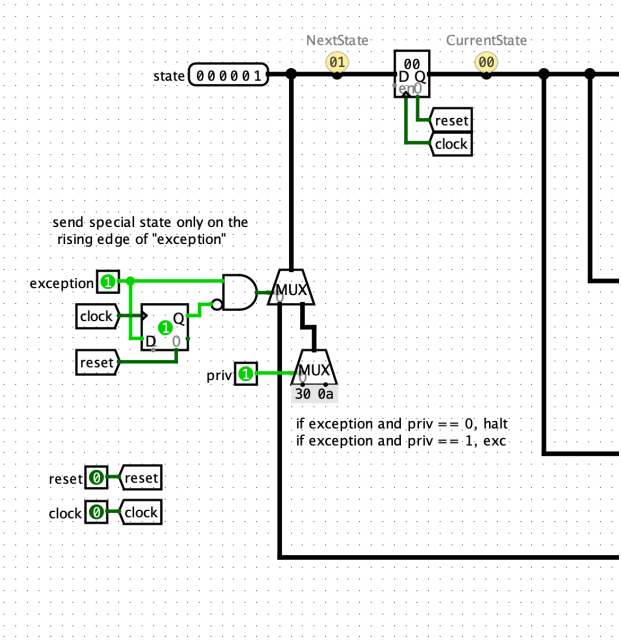

Figure 4: Exception extension to FSM

Testing

I used `exception.ys` to validate my circuit, choosing different instructions to test/uncomment each time. I tested different combinations of illegal `rA` and/or illegal `rB` registers, in both privileged and unprivileged mode.

1D.

The program outputs “Hi! Yes?” to TTY in kernel mode, throws a bad instruction exception while in user mode, then prints “Uh-oh!” in the exception trap at `0x200` before halting.

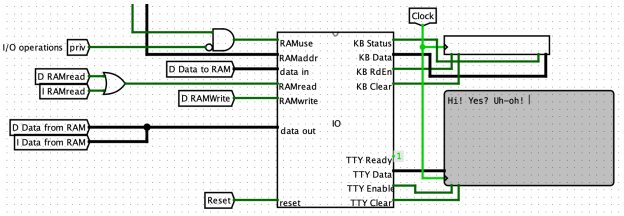

Figure 5: Output from `privdemo-F24.ys`

2.

I tested my MMU for the functionalities described in the rubric:

1. Allows permitted privileged reads/writes
2. Allows permitted unprivileged reads/writes
3. Throws exception if page is invalid
4. Throws exception if unprivileged code reads/writes a valid but forbidden address
5. RAM Use must be asserted for any read/write (added)

No.	P	Use (i)	R (i)	W (i)	V?	Un. W?	Un. R?	Use (o)	R (o)	W (o)	E
1	0	1	1	0	1	0	0	1	1	0	0
1	0	1	0	1	1	0	0	1	0	1	0
2	1	1	1	0	1	0	1	1	1	0	0
2	1	1	0	1	1	1	1	1	0	1	0
3	1	1	1	0	0	0	1	0	0	0	1
4	1	1	1	0	1	0	0	0	0	0	1
4	1	1	0	1	1	0	0	0	0	0	1
5	1	0	1	0	1	0	1	0	0	0	0