BPOS: Excercises for week 9

student: Dimitri Tabatadze (group: G3)

May 23, 2023

Solutions

1. (a) Since the need for 4 cycles comes from: i) translation for fetch, ii) instruction fetch, iii) translation for load/store, and iv) load/store, we can conclude, that step iii is not needed for the instructginons that don't perform load/store. This will change the pattern the bits $(E, phase)$ go through to this:

$$(0,0),(0,1),(1,1).$$

Where $h.phase$, obviously, is the register we used to split the fetch/execute phases into two subphases.

(b) We define the regular length $rle(i)$ of execution of instruction $I(d^i)$ by

$$rle(i) = \begin{cases} 2, & \neg mode(d^i), \\ 3, & mode(d^i). \end{cases}$$

We define the predicate $intd(i)$ stating that the execution of instruction $I(d^i)$ was interrupted by:

$$intd(i) \equiv \exists j < rle(i) : jisr(h^{t(i)+j}, eev_h^{t(i)+j}).$$

Then the length le(i) of the execution of the instruction is the number of cycles until the first interrupt in case there is one and the regular length of execution otherwise:

$$le(i) = \begin{cases} 1 + \min\{j | jisr(h^{t(i)+j}, eev_h^{t(i)+j})\}, & intd(i) \\ rle(i), & \neg intd(i). \end{cases}$$

Now we can define

$$t(i+1) = t(i) + le(i).$$

As before, interrupts are accumulated during execution of an instruction:

$$eev_{isa}^i = \bigvee_{t=t(i)}^{t(i+1)1} eev_h^t.$$

Now, we can use Lemma 130 to prove the correctness since this is very simmilar to what we have proven before.

2. (a) can be found in the book, page 379 through 380.
   (b) page 380

3. In any MIPS configuration d, two special purpose registers determine the region of memory $d.m$ whose content is interpreted as a single *pagetable*, namely

   - the page table origin register $d.pto$. It stores the byte address where the page table starts. This address must be aligned to word boundaries, i.e., we require
   - the page table length register $d.ptl$. It specifies the number of page table entries in the page table. Page table entries will contain four bytes, thus $d.ptl$ specifies the length of the page table measured in words.

   In user mode, `movg2s` can be used to set $d.pto$ to the start of the memory and $d.ptl$ to the maximum. This will enable access to the whole memory.

4.

5. (a) The whole code:

```
1  void writedisk(uint ppx, uint spx) {
2      int y;
```

   - set $spa(d) = spx$

```
3      writems(spx, hdbase + 4*K);
```

   - Load the data into the buffer

```
4      copyms(ppx*4*K, hdbase, k);
```

   - issue write access $cmsr(d) = 2_2 = 10$

```
5      writems(2, hdbase + 4*K + 4);
```

   - polling so that we can conclude that the buffer is copied to the swap memory.

```
6      y = 1;
7      while y != 0 {
8          y = readms(hdbase + 4*K + 4)
9      }
10 };
```

   (b) The correctness proof for this program has one new aspect: we have to argue when disk steps (signaling the completion of a disk access) can occur. Because idle disks do not make steps, there is no disk step before the access is started with the write to the command and

status register. Using C+A semantics we are looking for execution of two writems procedures in a configuration $y^{j_0} = (d^{j_0}, c^{j_0})$ satisfying

$$spa(d^{j_0}) = va(spx, c)_{28}$$
$$cmsr(d^{j_0}) = 2_{32}$$

hence

$$hdbusy(d^{j_0}).$$

The ISA portion of the extended C+A computation is normal. Thus the disk step signalling the completion of the disk access is reordered to occur before an assembly instruction accessing the disk. During the polling the only such instruction is the load instruction

```
lw  3 2 0
```

belonging to the loop body, which samples the command and status register. For $j \geq 1$ let $z(j)$ be extended C+A configuration, in which this lw instruction is executed in iteration j of the loop. We denote the corresponding instruction count as

$$x(j) = z(j).ic.$$

If the disk is found to be busy another iteration of the loop follows. Thus the software condition for the liveness of disks in reordered computations from Sect. 14.3.6 of the book is fulfilled. We can apply Lemma 137 and conclude that for some iteration j_k of the loop the disk is idle in ISA configuration $z(j_k).d$ and the loop is left after a successful disk operation. That the desired data is read from the disk follows from the ISA semantics for disks.