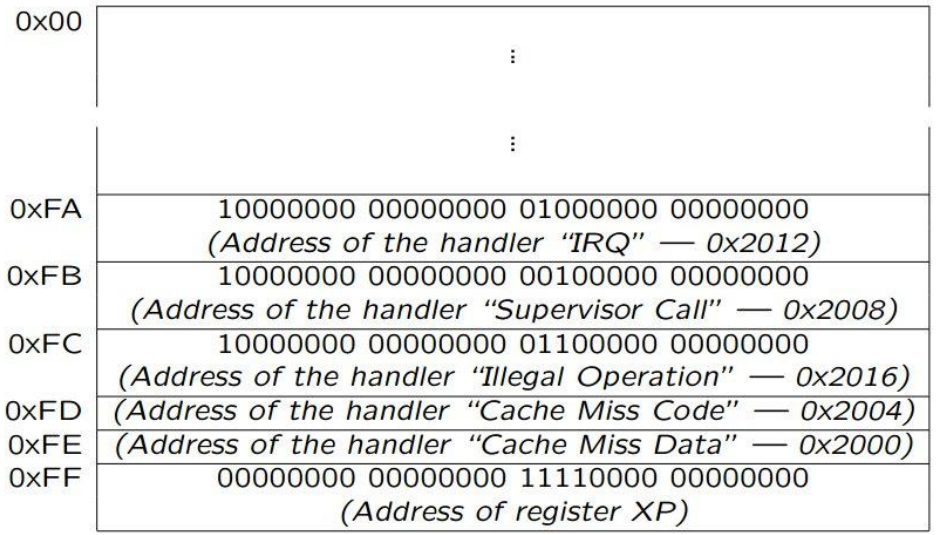
# Computation structures
# Tutorial 4: μ-code for ULg03

# ULg02 - constant ROM and XP register

- For handling exceptions, ULg02 introduces **the XP (R30) register** in which is stored the user program address at which the execution must return after the system have finished executed an exception handler.
- ULg02 introduces a control ROM storing useful constants:
  - Address of XP
  - Address of exception handlers

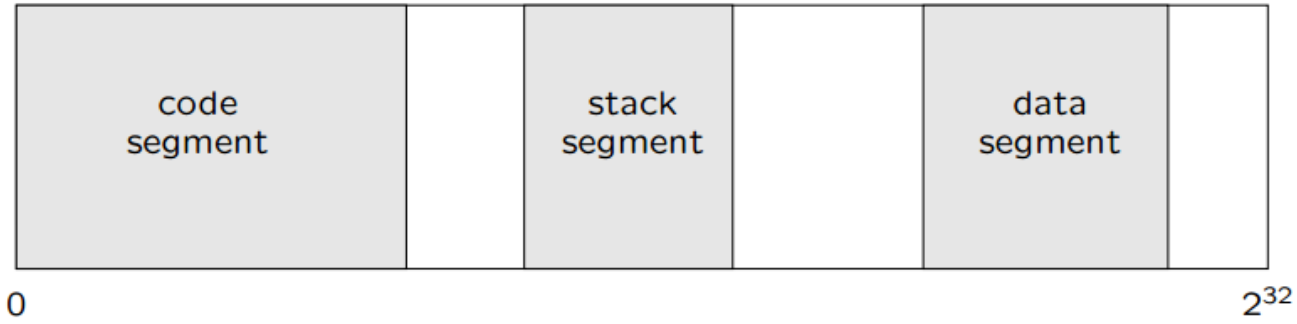


# ULg03, introducing virtual memory

- **Virtual memory**: user programs have virtually access to a large contiguous space of dynamic memory (larger than the actual amount of available physical memory). In practice, the **memory is fragmented into pages** which can be stored on disk.
- **Goal**:
  - Make it easier to have several processes
  - A process cannot mess with the memory of other processes, nor the system memory
  - More memory available than actual amount of available physical memory
  - No need for reserving a fixed amount of memory for processes execution

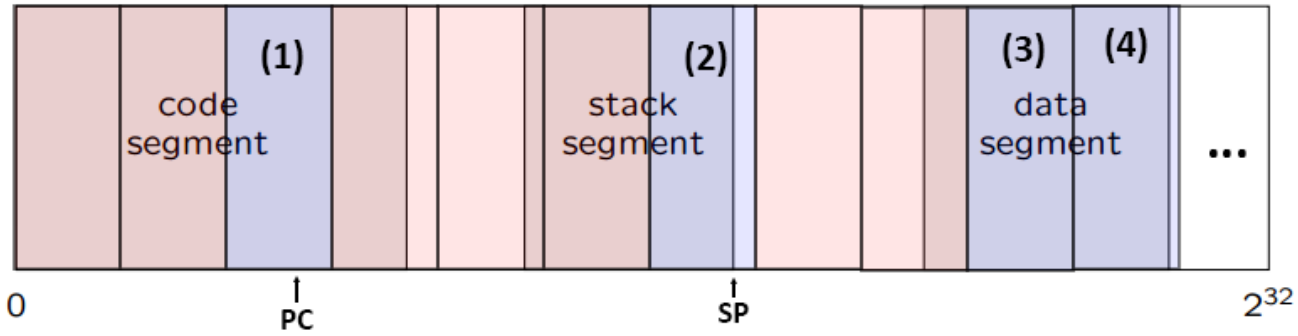
# Virtual memory – how it works

Virtual memory view of a user program



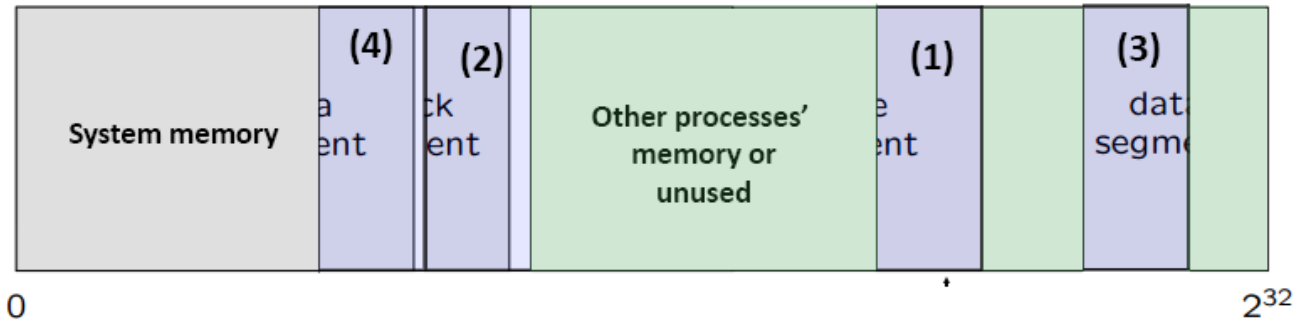

**System** operates in **physical memory**
**User programs** operate in **virtual memory**

Splitted in pages




A **page** is a fixed-size set of contiguous memory locations that can contain anything (code, stack, data,…)

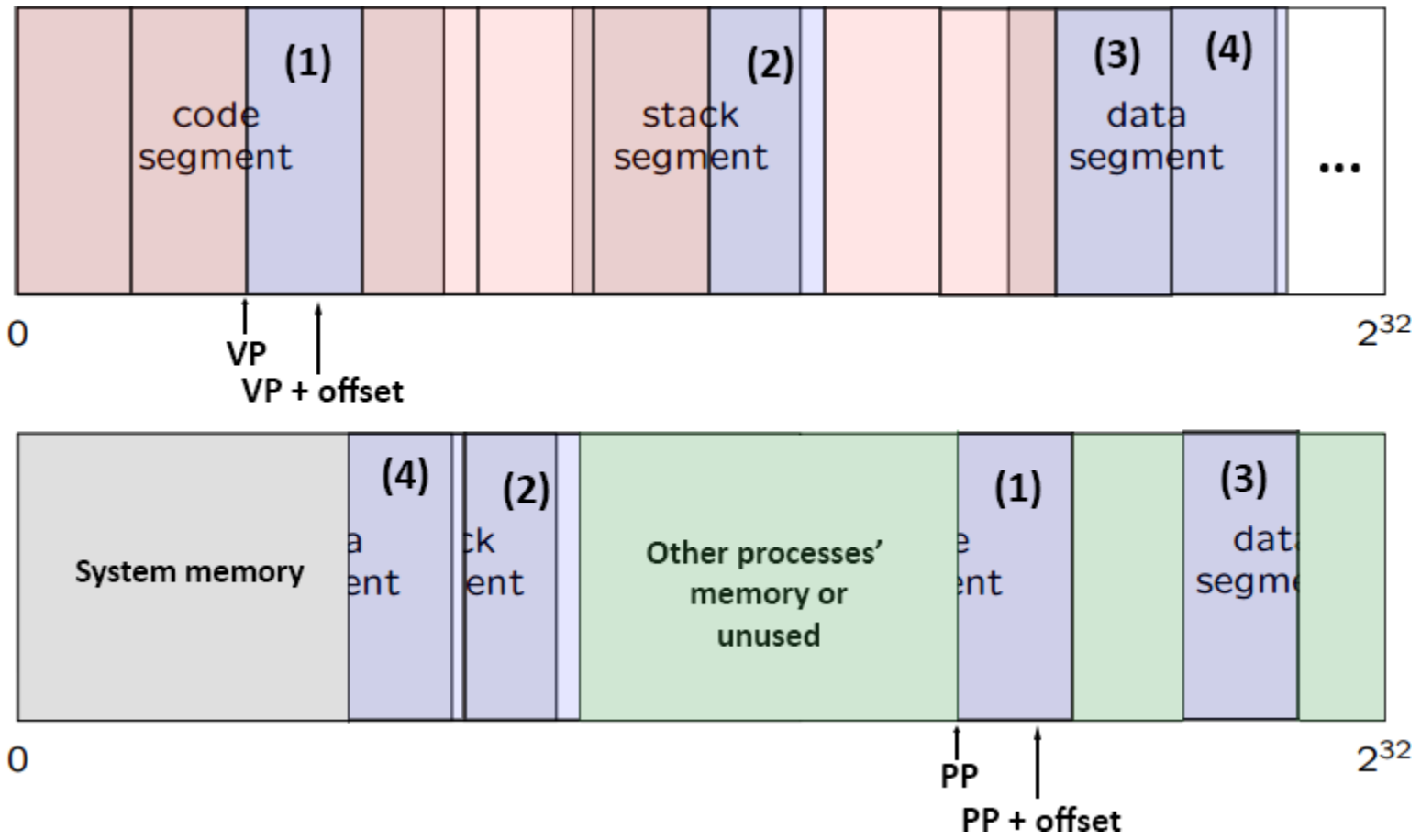
Physical (actual) memory




Useful pages are stored in memory while other can be stored on disk ⇒ virtually more memory available than actual physical memory

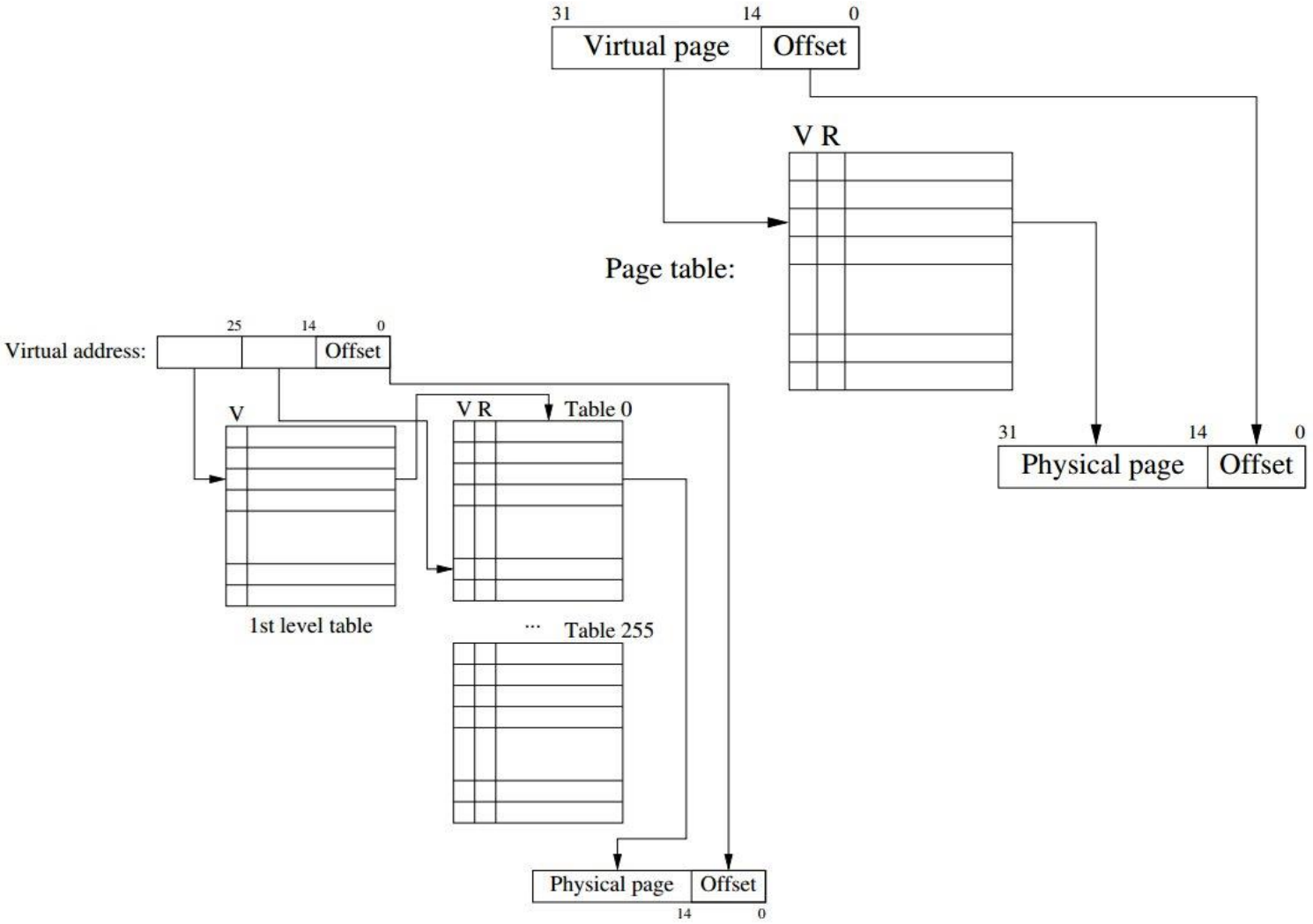
# Virtual memory – how it works

- **Virtual address**: an address in the virtual space
- **Physical address**: the address in the actual memory
- All addresses have two components:
  - Page address: VP (virtual page) or PP/SP (physical page/system page)
  - Offset: the address offset in the page



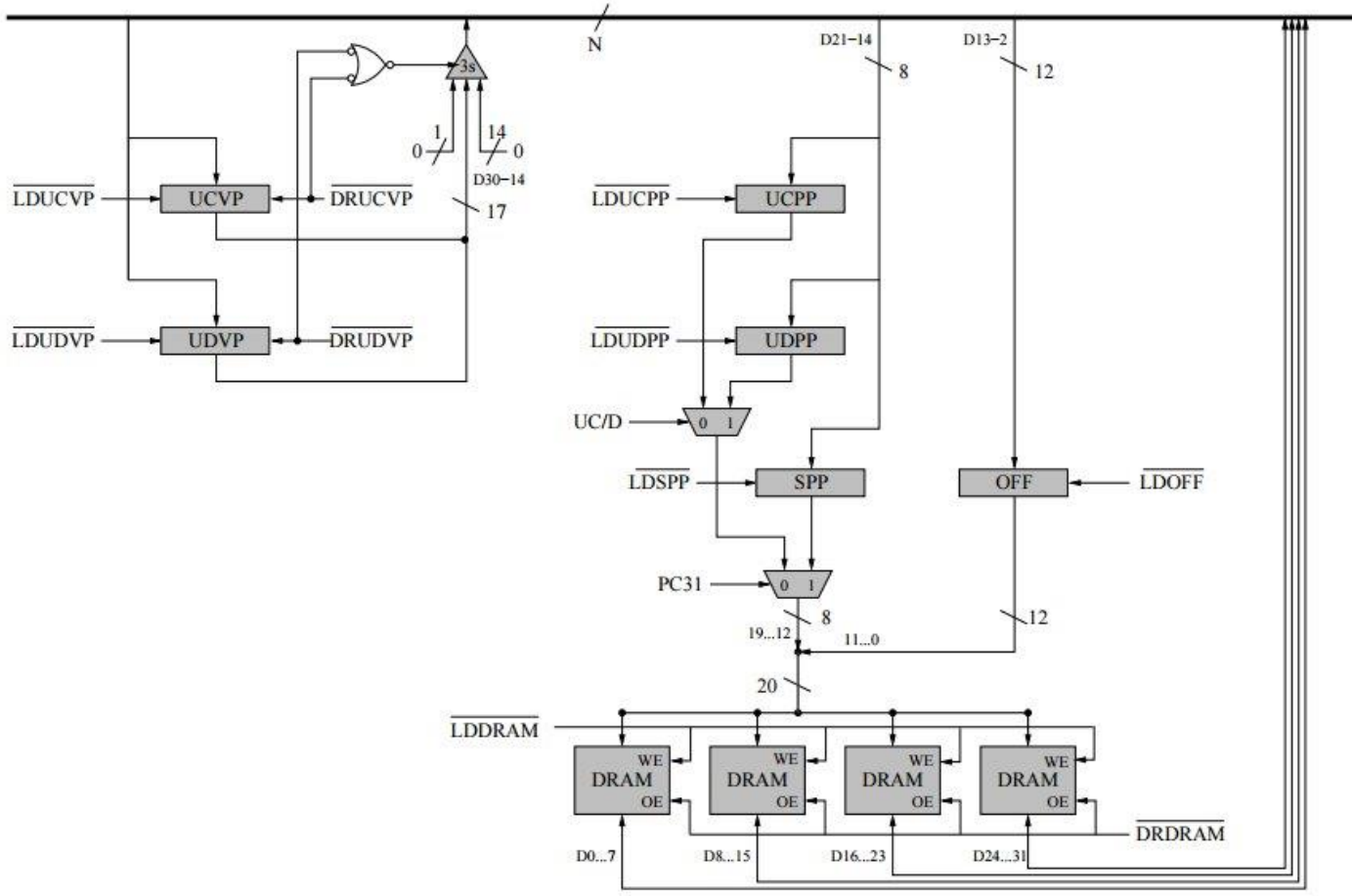
# Virtual memory – page table

- The system translates virtual pages (VP) into physical pages (PP) using a **page table**
- The page table is stored in the system memory
- Sometimes uses a two-level table for storage efficiency
- Need to access memory for translating virtual addresses ⇒ **very inefficient**



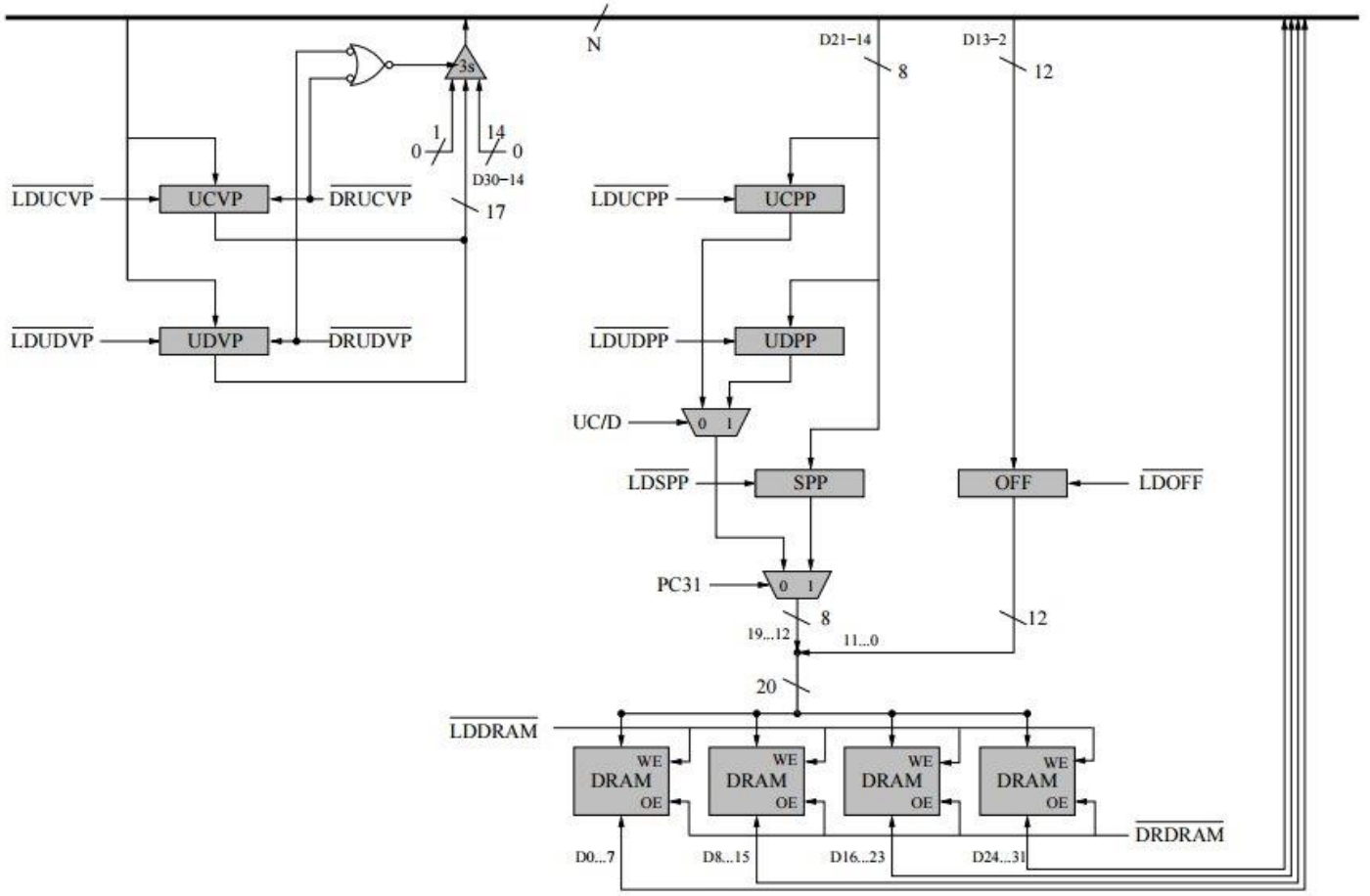
# Virtual memory – optimize look-up with caching

- **Locality principle**: accesses that are time-wise close are often address-wise close
- We can store PP, the result of the last translation of VP, in a register because the next translation will probably yield the same PP !
- Frequent memory accesses:
  - **Code access**: instructions
  - **Data access**: actual data or stack
- Both types of accesses are cached



# Virtual memory – optimize look-up with caching

- In user mode:
  - **U*VP and U*PP** contains respectively virtual page address and translated physical address
  - **C**ode: U**C**VP and U**C**PP
  - **D**ata: U**D**VP and U**D**PP
  - UC/D is a flag allowing to choose the cache among those two
- In SVR mode (PC31 = 1), uses physical memory:
  - SPP: System Physical Page
- In both cases, the offset is stored in register OFF



# Virtual memory – cache hit and miss

- **Cache hit**: if the cache register contains the expected address.
- **Cache miss**: if the cache register doesn’t contain the expected address.
- Cache miss triggers a « Cache miss code » or a « Cache miss data » **exception** that is handled at the $\mu$-code level.
- The handlers of those exceptions update the corresponding cache with the expected address

# Virtual memory – a cache miss data handler

XP – 4 for re-executing the interrupted instruction
(only for cache miss data, not for cache miss code)

```
h_stub:  SUBC(XP, 4, XP)        | plan to reexecute the instruction
                                | that has been suspended
         ST(r0, User, r31)      | save
         ST(r1, User+4, r31)
              . . .
         ST(r30, User+30*4)
         CMOVE(KStack, SP)      | Load the system SP
         RDUDVP(r1)             | the virtual page address to be
                                | translated
         PUSH(r1)               | pass it as argument

         BR(CMHandler,LP)       | call the  Handler
         DEALLOCATE(1)          | remove the argument from the stack
         WRUDPP(r0)             | install the returned value
         LD(r31, User, r0)      | restore
         LD(r31, User+4, r1)
         LD(r31, User+30*4, r30)
         JMP(XP)                | return to application
```

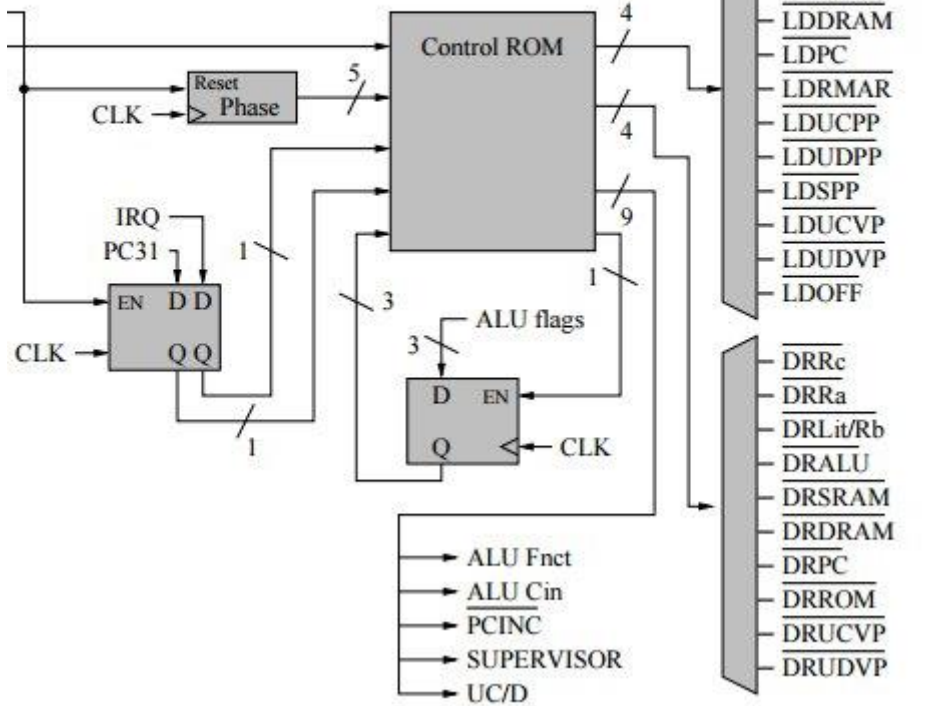
Update physical page address UDPP

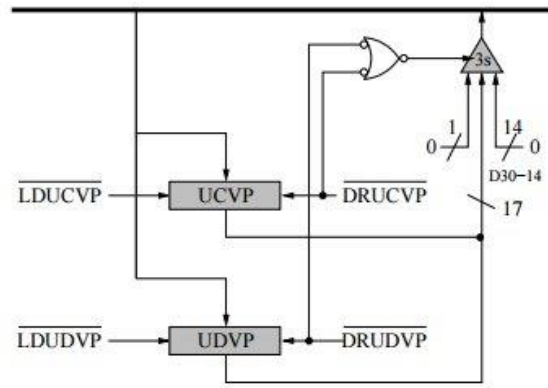
Go back to the interrupted instruction (of which the address is stored in XP)

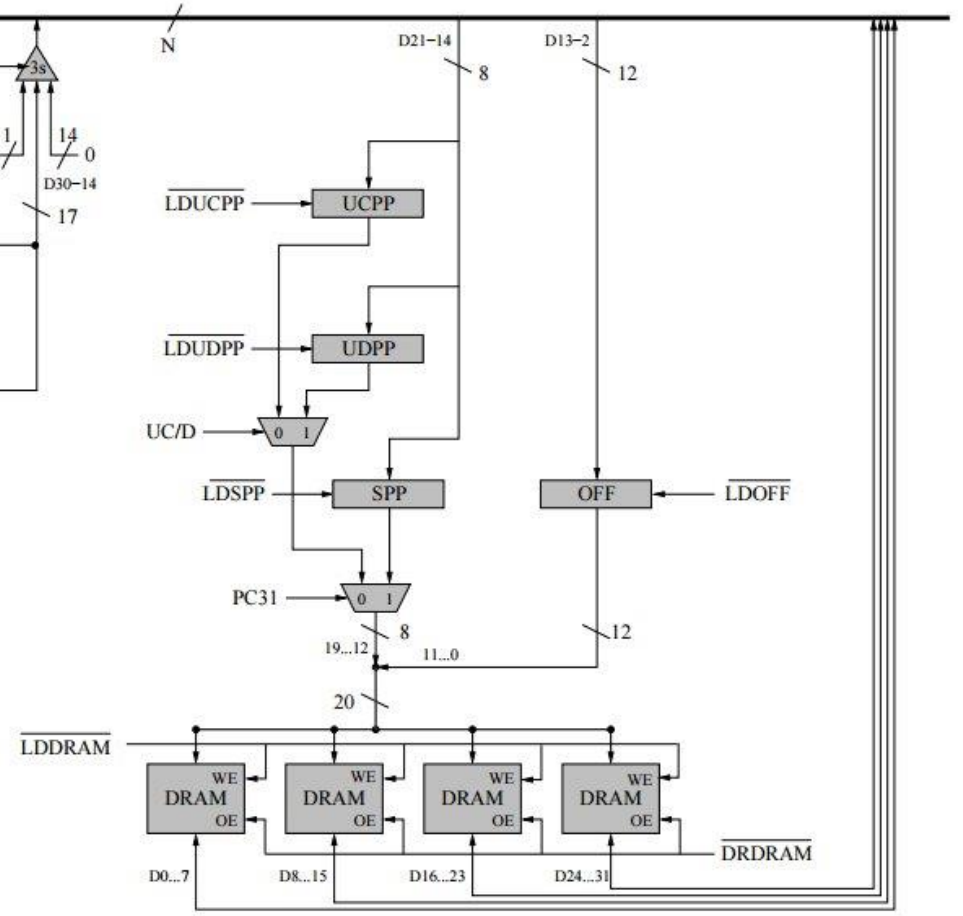
| | |
|---|---|
| 0x00 | ⋮ |
| | ⋮ |
| 0xFA | 10000000 00000000 01000000 00000000<br>*(Address of the handler "IRQ" — 0x2012)* |
| 0xFB | 10000000 00000000 00100000 00000000<br>*(Address of the handler "Supervisor Call" — 0x2008)* |
| 0xFC | 10000000 00000000 01100000 00000000<br>*(Address of the handler "Illegal Operation" — 0x2016)* |
| 0xFD | *(Address of the handler "Cache Miss Code" — 0x2004)* |
| 0xFE | *(Address of the handler "Cache Miss Data" — 0x2000)* |
| 0xFF | 00000000 00000000 11110000 00000000<br>*(Address of register XP)* |

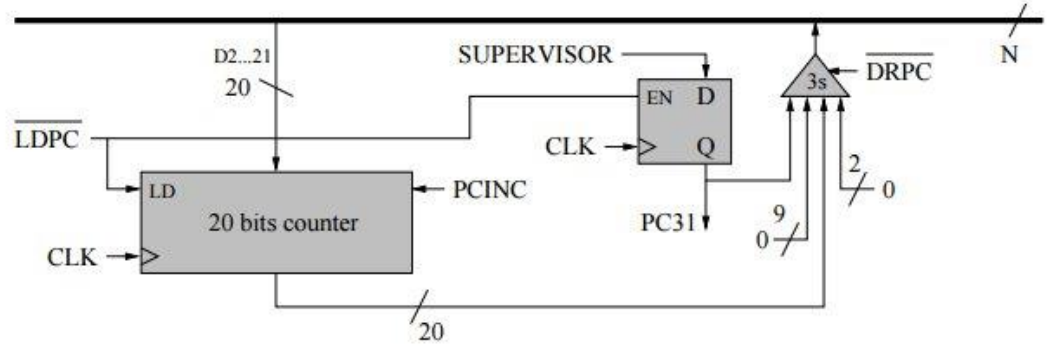
| Control Inputs | | | | | | Output |
|---|---|---|---|---|---|---|
| F3 | F2 | F1 | F0 | C | M | |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0xffffffff |
| 1 | 1 | 1 | 1 | 1 | 1 | A |
| 1 | 0 | 1 | 0 | 1 | 1 | B |
| 1 | 1 | 1 | 1 | 1 | 0 | A-1 |
| 0 | 0 | 0 | 0 | 0 | 0 | A+1 |
| 1 | 0 | 0 | 1 | 1 | 0 | A+B |
| 0 | 1 | 1 | 0 | 0 | 0 | A-B |
| 1 | 0 | 1 | 1 | 1 | 1 | A and B |
| 1 | 1 | 1 | 0 | 1 | 1 | A or B |
| 0 | 1 | 1 | 0 | 1 | 1 | A xor B |
| 0 | 0 | 0 | 0 | 1 | 1 | not A |
| 1 | 1 | 0 | 0 | 1 | 0 | A+A |
| 0 | 1 | 1 | 0 | 1 | 0 | A-B-1 |
| 1 | 0 | 0 | 1 | x | 1 | not (A xor B) |
| 0 | 1 | 0 | 0 | x | 1 | not (A and B) |

*flags:*
*E=1 <=> Alu out = 0xffffffff*
*C : negated carry out*
*N= most sign. Alu out: a31*

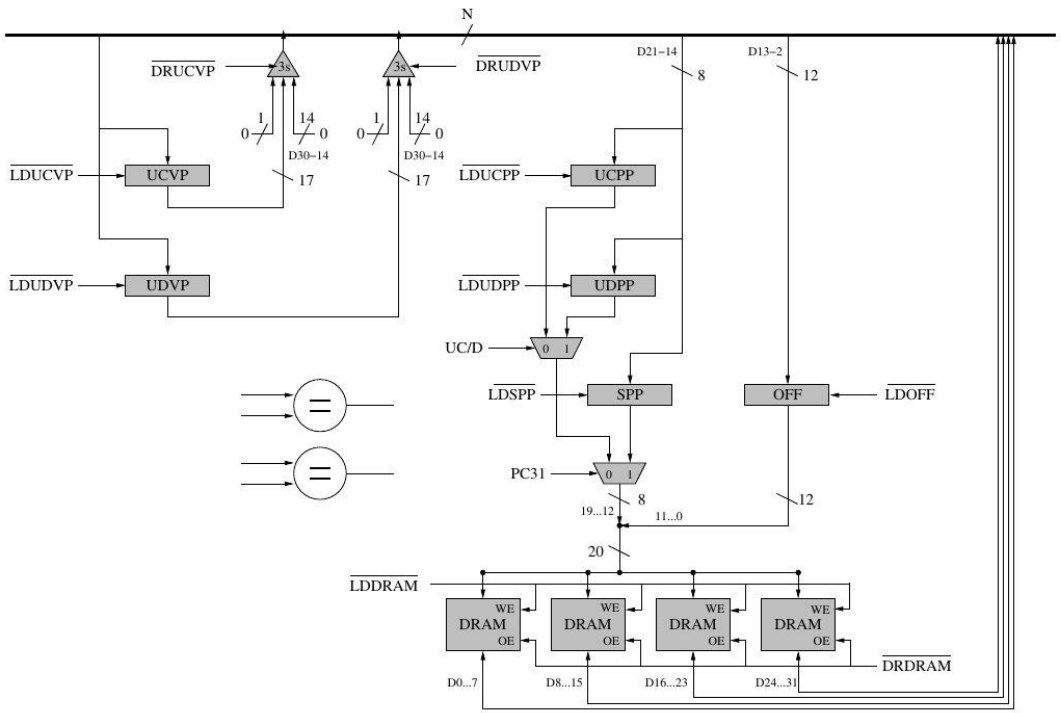


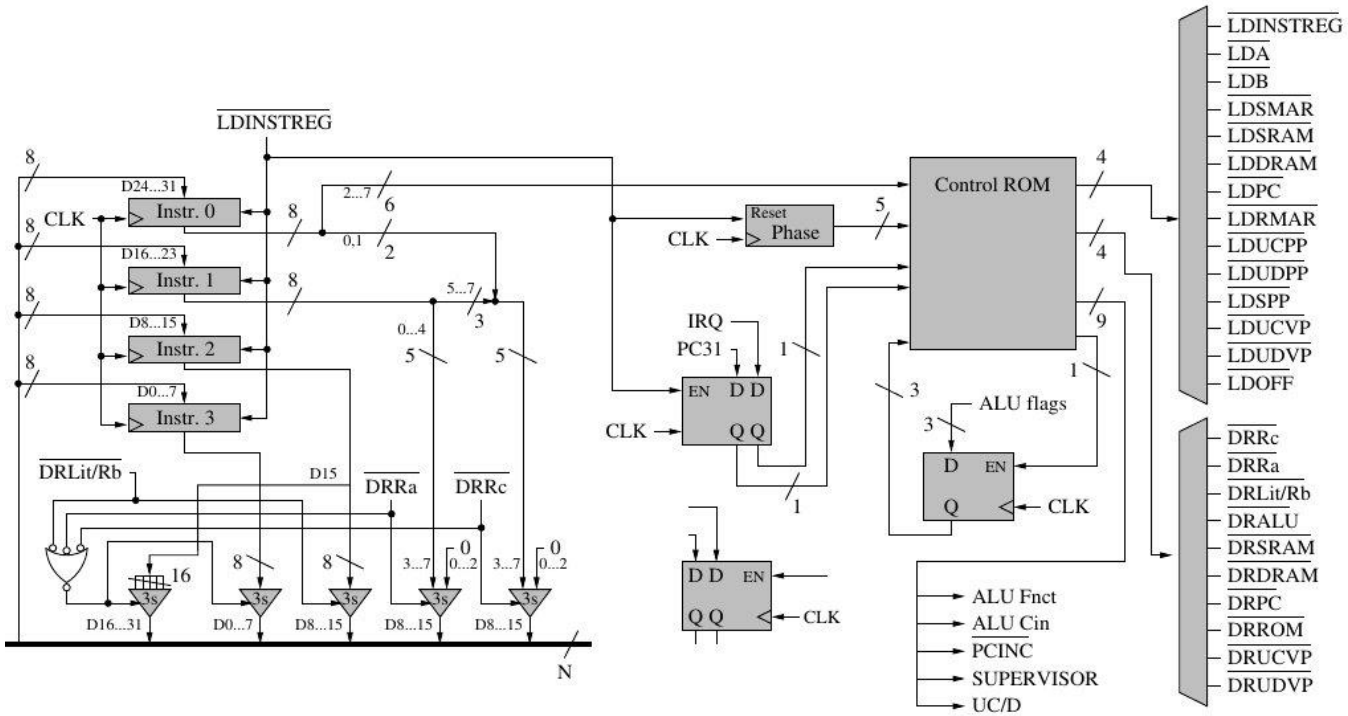



# Exercise 2 - January 2017

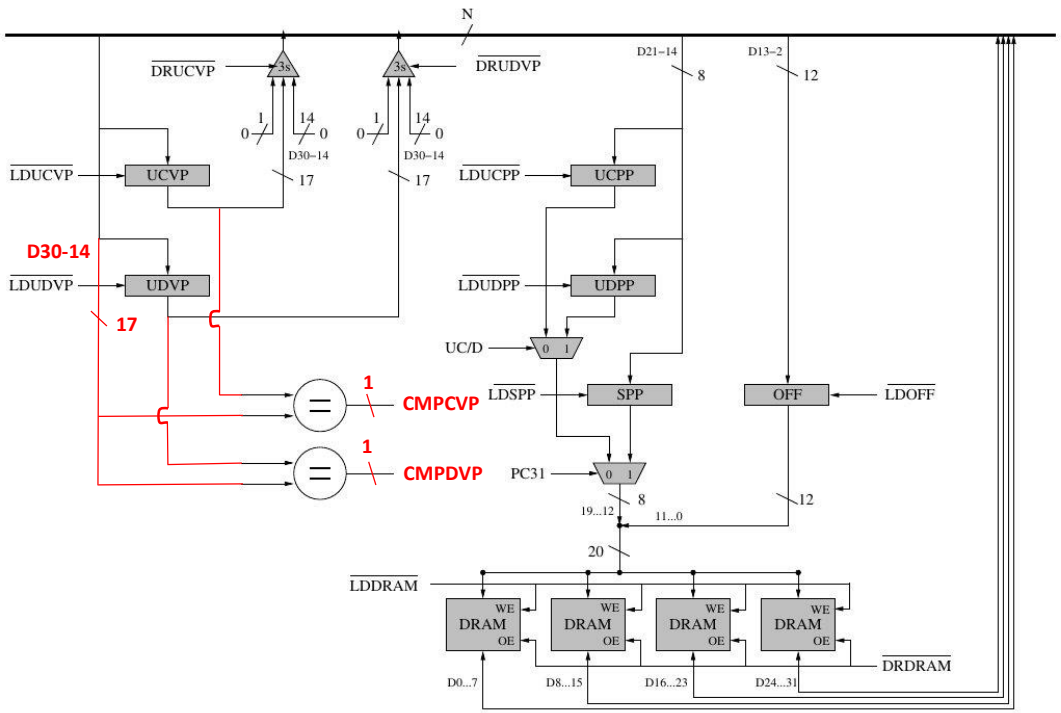



(b) Virtual RAM (*hint*: notice the new unconnected comparators and the new output wiring of UCVP and UDVP, which are no longer 3 state)

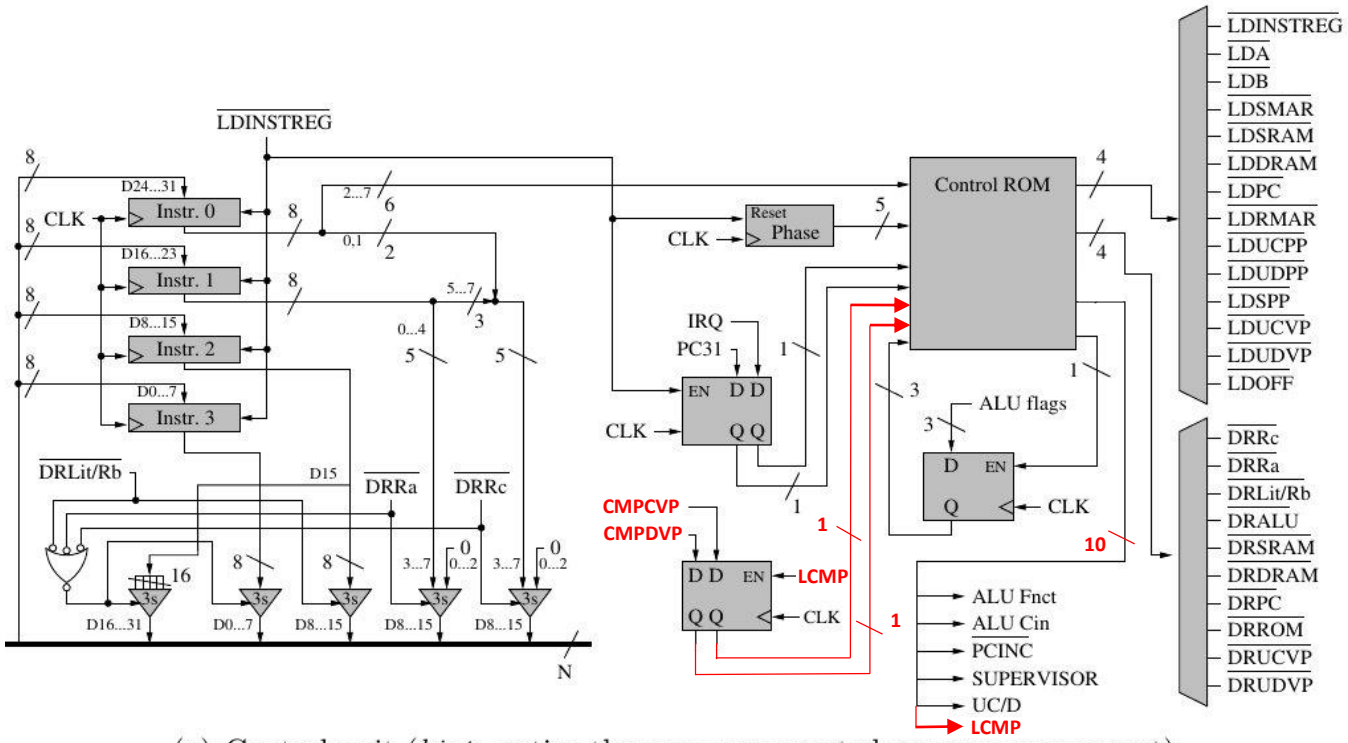



(a) Control unit (*hint*: notice the new unconnected memory component)

# Exercise 2(a) - January 2017




(b) Virtual RAM (*hint*: notice the new unconnected comparators and the new output wiring of UCVP and UDVP, which are no longer 3 state)




(a) Control unit (*hint*: notice the new unconnected memory component)

# Exercise 2(a) - January 2017

Optimized
(with hardware comparison)

| Cond | ALU | DR | LD | Sig | |
|---|---|---|---|---|---|
| | | Ra | SMAR | | 0 |
| | | SRAM | A | LCMP | 1 |
| CMPDVP = 0 | 0xffffffff | ALU | A | | 2 |
| | A | ALU | RMAR | | 3 |
| | | ROM | SMAR | | 4 |
| | | PC | SRAM | | 5 |
| | A-1 | ALU | RMAR | | 6 |
| | | ROM | PC | SVR | 7 |
| | | PC | SPP | | 8 |
| | | PC | OFF | PC+ | 9 |
| | | DRAM | INSTREG | | 10 |
| CMPDVP = 1 | A | ALU | OFF | | 2 |
| | | DRAM | B | UC/D | 3 |
| | | Rb | SMAR | | 4 |
| | | SRAM | A | | 5 |
| | | Rc | SMAR | | 6 |
| | A-B | ALU | A | LF | 7 |
| N = 0 | 0 | ALU | SRAM | | 8 |
| N = 1 | B | ALU | SRAM | | 8 |
| | A | ALU | A | | 9 |
| | | PC | OFF | LCMP | 10 |
| CMPCVP = 0 | 0xffffffff | ALU | A | | 11 |
| | A | ALU | RMAR | | 12 |
| | | ROM | SMAR | | 13 |
| | | PC | SRAM | | 14 |
| | A-1 | ALU | A | | 15 |
| | A-1 | ALU | RMAR | | 16 |
| | | ROM | PC | SVR | 17 |
| | | PC | SPP | | 18 |
| | | PC | OFF | PC+ | 19 |
| | | DRAM | INSTREG | | 20 |
| CMPCVP = 1 | | DRAM | INSTREG | PC+ | 11 |

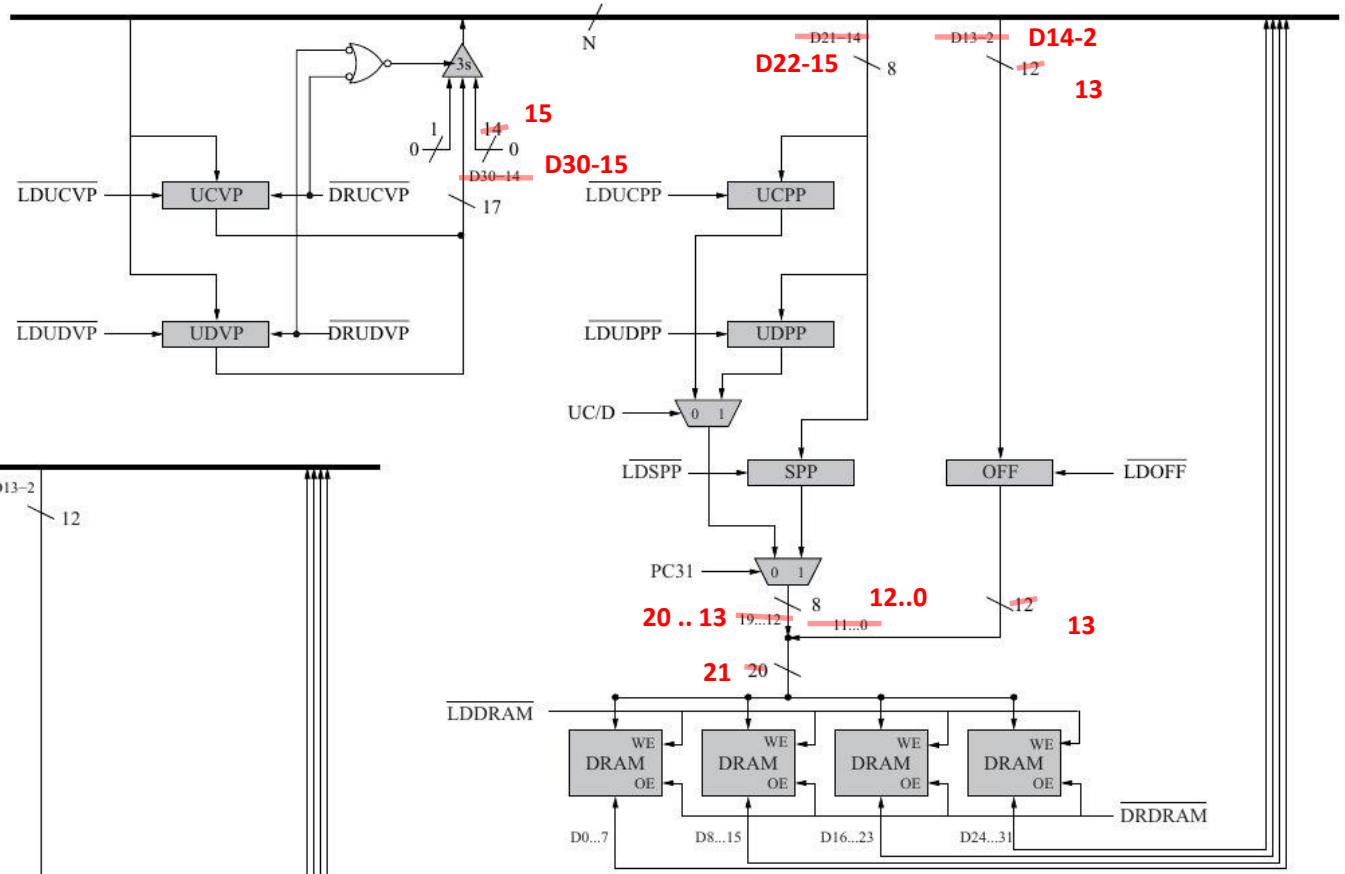
Basic
(without hardware comparison)

| Cond | ALU | DR | LD | Sig | |
|---|---|---|---|---|---|
| | | UDVP | A | | 0 |
| | | Ra | SMAR | | 1 |
| | | SRAM | OFF | | 2 |
| | | SRAM | UDVP | | 3 |
| | | UDVP | B | | 4 |
| | !(A^B) | ALU | A | LF | 5 |
| E = 0 | 0xffffffff | ALU | A | | 6 |
| | A | ALU | RMAR | | 7 |
| | | ROM | SMAR | | 8 |
| | | PC | SRAM | | 9 |
| | A - 1 | ALU | RMAR | | 10 |
| | | ROM | PC | SVR | 11 |
| | | PC | SPP | | 12 |
| | | PC | OFF | PC+ | 13 |
| | | DRAM | INSTREG | | 14 |
| E = 1 | A | ALU | A | | 6 |
| | A | ALU | A | | 7 |
| | A | ALU | A | | 8 |
| | A | ALU | A | | 9 |
| | | DRAM | B | UC/D | 10 |
| | | Rb | SMAR | | 11 |
| | | SRAM | A | | 12 |
| | | Rc | SMAR | | 13 |
| | A-B | ALU | A | LF | 14 |
| N = 0 | 0 | ALU | SRAM | | 15 |
| N = 1 | B | ALU | SRAM | | 15 |
| | | UCVP | A | | 16 |
| | | PC | OFF | | 17 |
| | | PC | UCVP | | 18 |
| | | UCVP | B | | 19 |
| | !(A ^ B) | ALU | A | LF | 20 |
| E = 0 | 0xffffffff | ALU | A | | 21 |
| | A | ALU | RMAR | | 22 |
| | | ROM | SMAR | | 23 |
| | | PC | SRAM | | 24 |
| | A-1 | ALU | A | | 25 |
| | A-1 | ALU | RMAR | | 26 |
| | | ROM | PC | SVR | 27 |
| | | PC | SPP | | 28 |
| | | PC | OFF | PC+ | 29 |
| | | DRAM | INSTREG | | 30 |
| E = 1 | | DRAM | INSTREG | PC+ | 21 |

- Optimized version almost two times faster when there is no cache miss (11 phases instead of 21)
- Notice the padding in the basic version (not necessary in optimized version as we use different memory components for the flags)

# Exercise 3

Doubling the page size




Doubling the number of pages