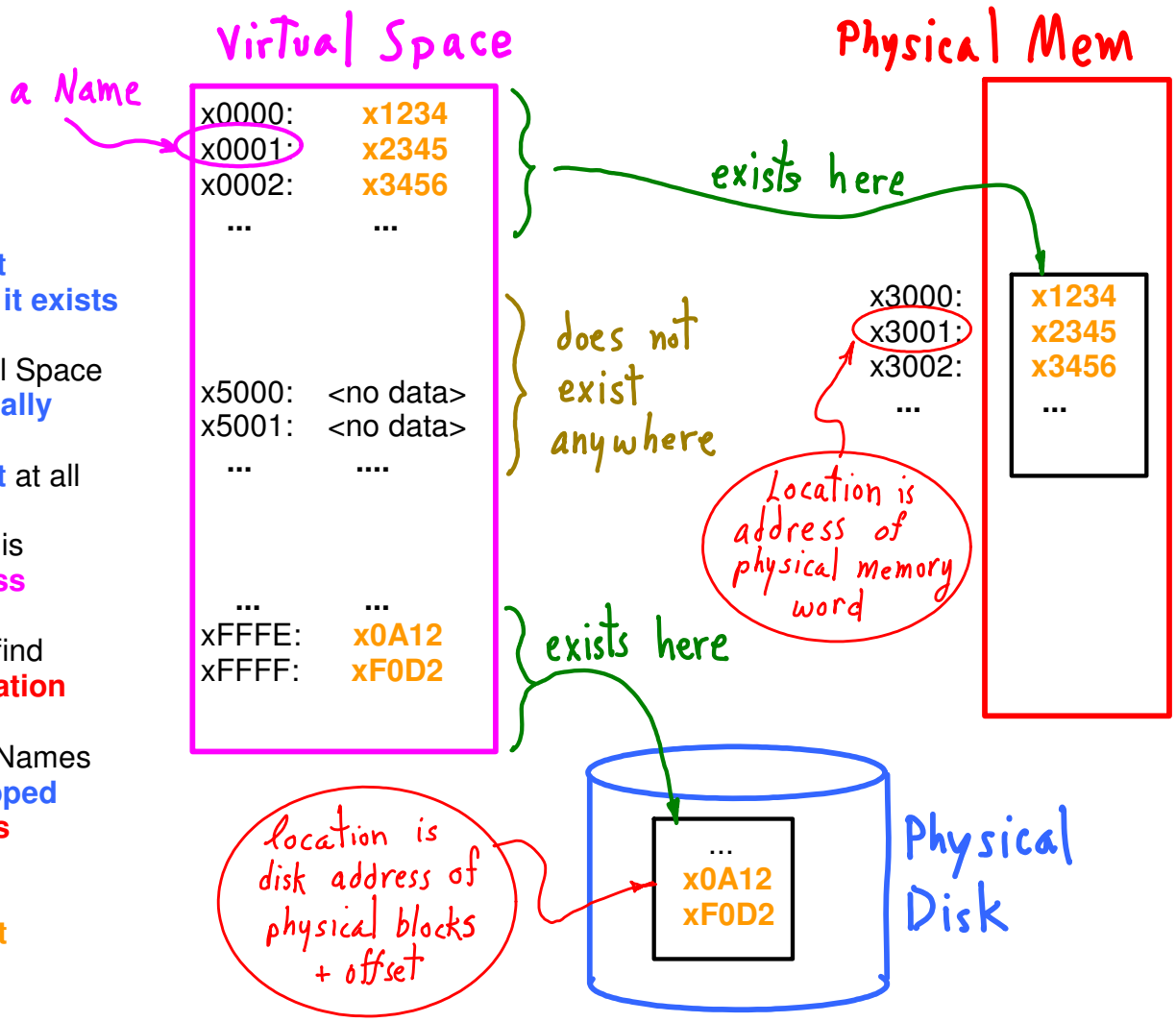
Lec-3d-VMsummary

Let's use LC3
--- 16-bit words (2B)
--- 2B-addressable memory
--- 16-bit physical addresses

Add:
--- 16-bit virtual address space
--- 4k word pages
--- page number = 1st hex digit of Virtual Address
--- frame number = 1st hex digit of Physical Address



MAPPING

Virtual Space
--- does not exist
--- we "imagine" it exists

Content of Virtual Space
--- **exists, physically**
or
--- **does not exist** at all

Content's Name is
--- **Virtual Address**

Name is used to find
--- **Content's location**

Processor uses Names
--- which are **mapped to locations**

Location is used
--- **to get Content**

Aside: General Extended Names,
--- Process ID
--- Thread ID
--- Physical Disk Address (Head, Cylinder, Sector)
--- Logical Disk Sector
--- File System Name (e.g., "/bin/rm")
--- User/Owner Name
--- Network address
--- Host/Domain name
--- Distributed OS object name
etc.

Disk block = 2^8 words = 256 words = 512B

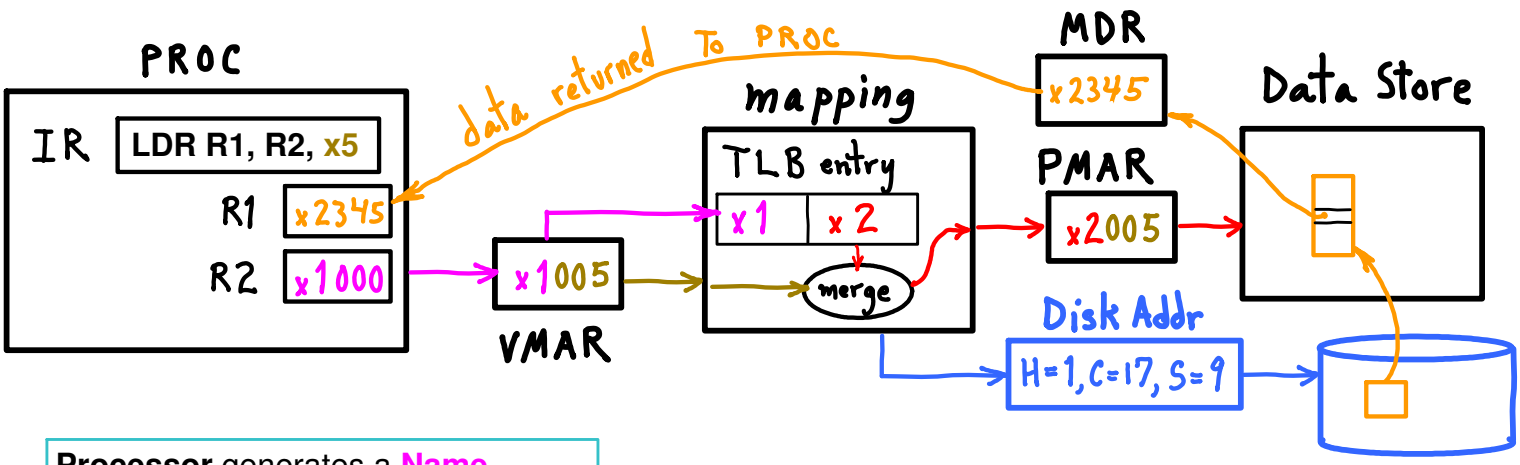
Page = 2^12 words = 4k words = 8kB
= 2^4 blocks = 16 blocks

Memory = 2^16 words = 64k words = 128kB
= 2^8 blocks = 256 blocks
= 2^4 Pages = 16 Pages

Page Table = 16 entries (PTEs)

Disk size = 2^8 pages = 256 pages = 2MB

Logical Process



Processor generates a **Name**
--- a **Virtual Address**
--- **sent to VMAR**, e.g.,

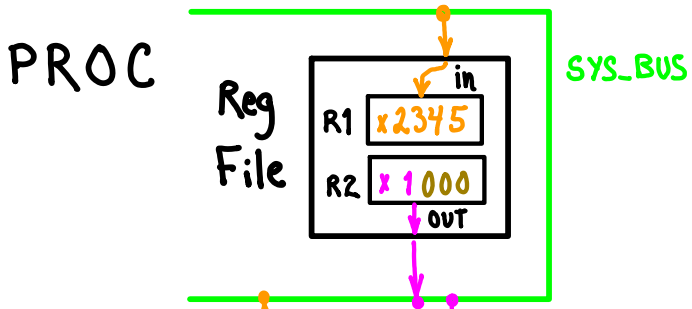
LDR R1, R2, x5

generates the **VAddress x1000 + x5**

Name is **translated** to **location**:
--- **physical memory address**
caches: yet **another name**
memory: a **physical location**
OR
--- **physical disk address**

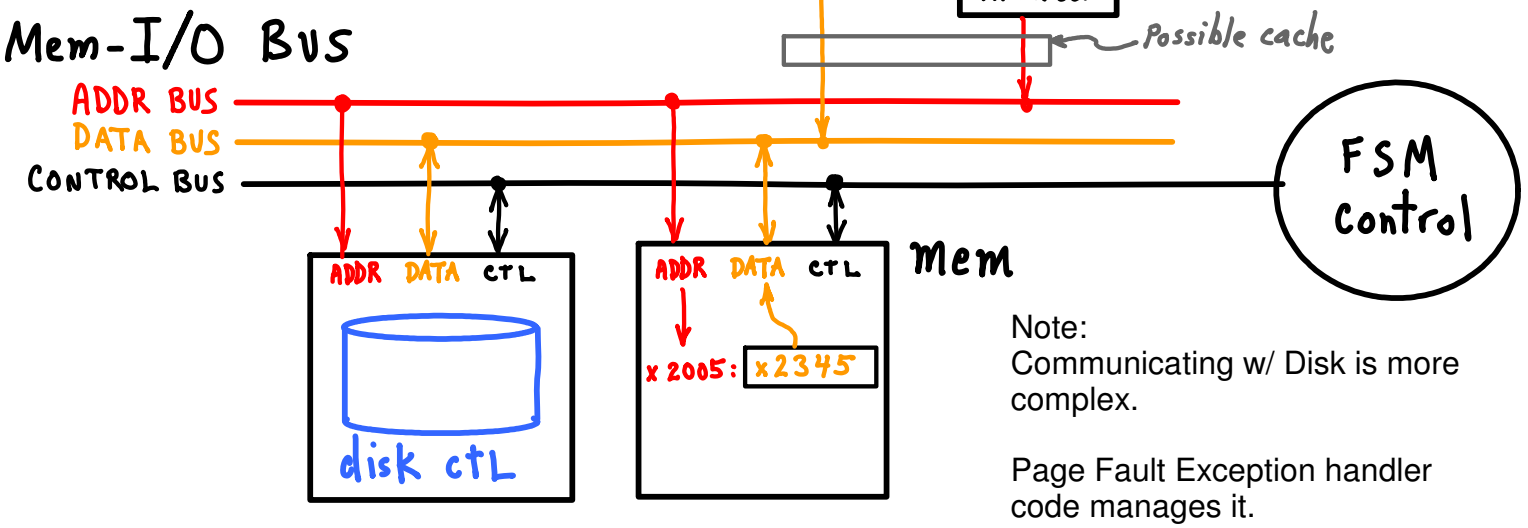
Data returned
--- via **Disk**
--- via **Memory**
--- via **Cache**
--- via **MDR**

Physical Process



R2 ===> SYS_BUS
===> MAR ===> Map
===> ADDR_BUS
===> Mem.ADDR (address decode)

Mem[x3001] ===> DATA_BUS
===> MDR
===> SYS_BUS
===> RegFile.in ===> R1



Note:
Communicating w/ Disk is more complex.

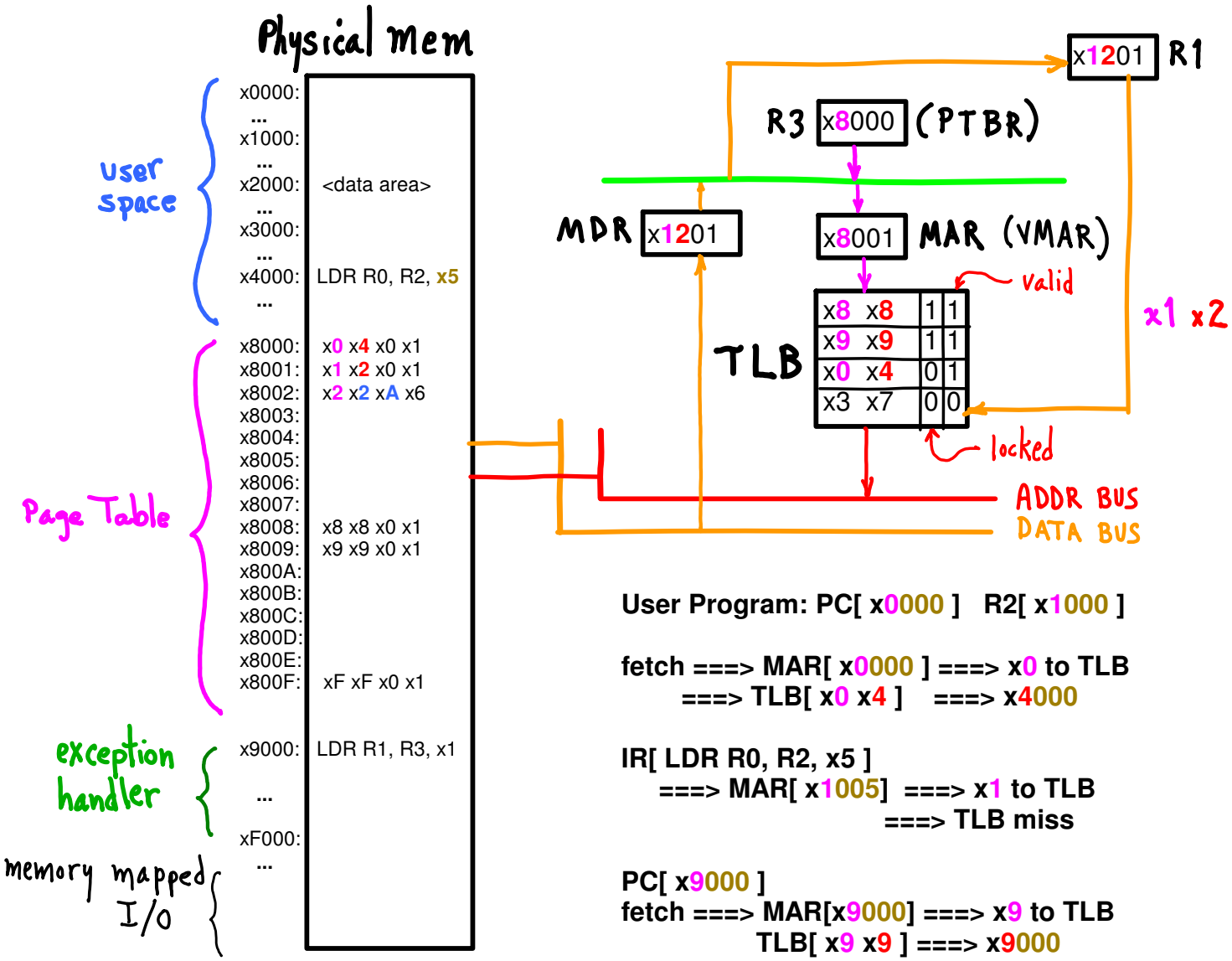
Page Fault Exception handler code manages it.

Page Tables

TLB is a **small cache**.

[P# : F#] not in TLB? ===> TLB miss exception
--- Jump to TLB Exception Handler code

--- Exeception Handler code:
map (PT) is in memory
--- get PTE, i.e., [P# : F#]
--- load TLB
--- restart instruction



User Program: PC[x0000] R2[x1000]

fetch ===> MAR[x0000] ===> x0 to TLB
===> TLB[x0 x4] ===> x4000

IR[LDR R0, R2, x5]
===> MAR[x1005] ===> x1 to TLB
===> TLB miss

PC[x9000]
fetch ===> MAR[x9000] ===> x9 to TLB
TLB[x9 x9] ===> x9000

IR[LDR R1, R3, x1]
===> MAR[x8001] ===> x8 to TLB
===> TLB[x8 x8] ===> x8001

MDR[x1 x2 x0 x1] ===> R1
TLB[x3 x7] <=== x1 x2]

PC[x0000] (restart instruction)

Page Table Entry, page in memory

1101 0011 0000 0001

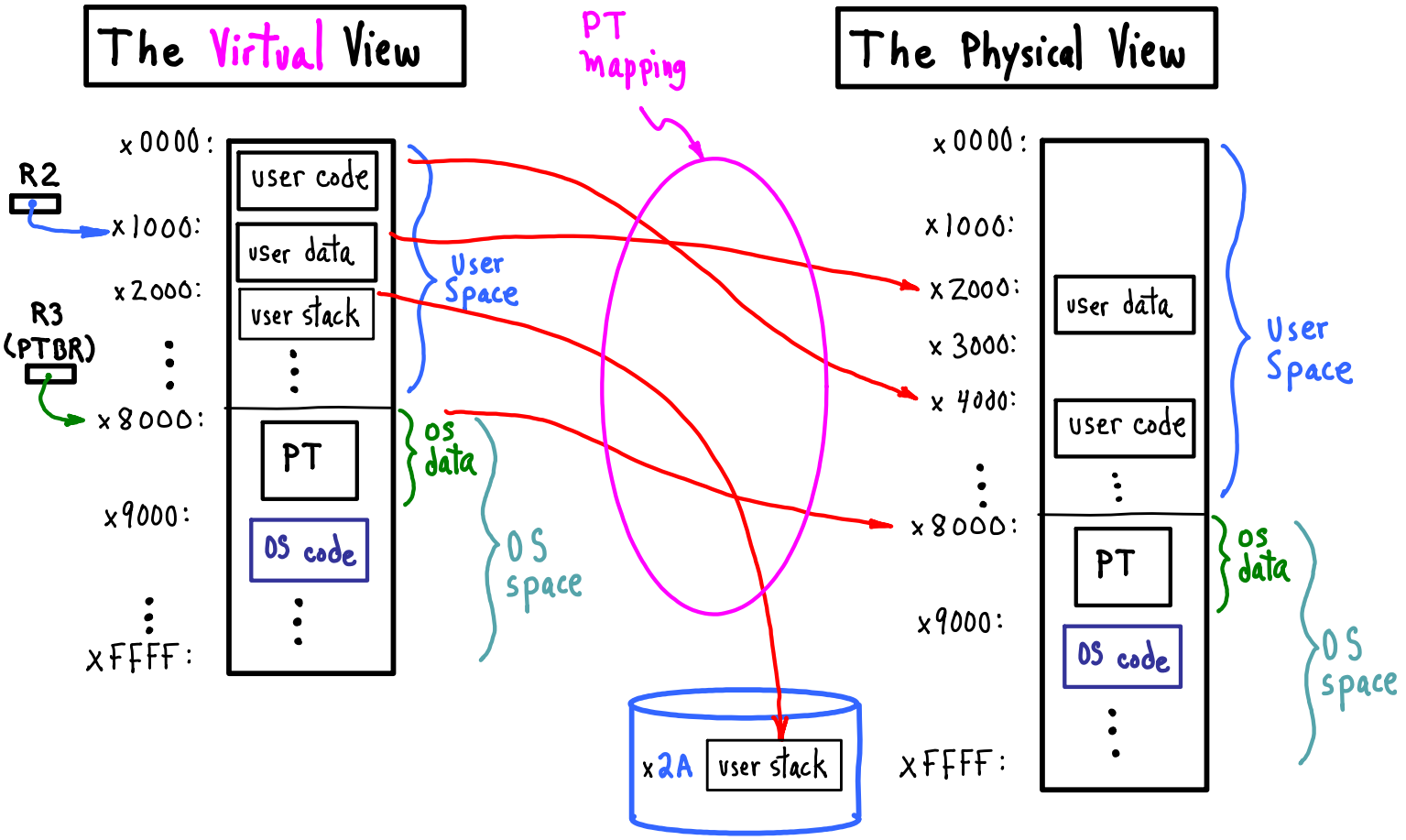
P# F# PID? m, locked, a? Present in memory

Page Table Entry, page not in memory

1101 0110 1011 0000 not in memory

Disk page # (Translated to Head, cylinder, sector disk addresses)

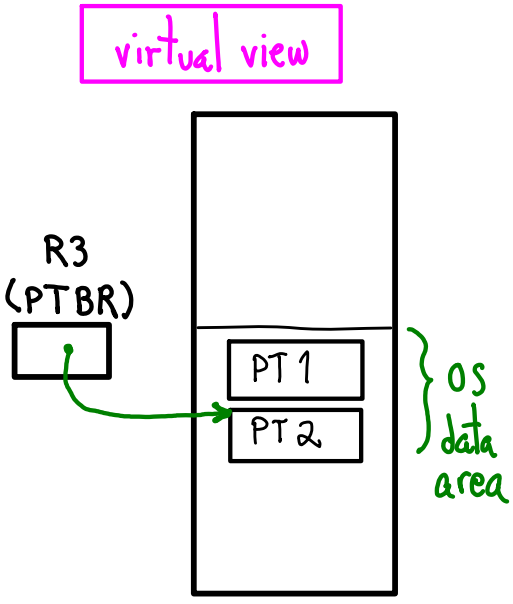
Better use of PTE bits: Don't need P# in PTE, P# is in VMAR, P# is PT index. (STill needed in TLB).

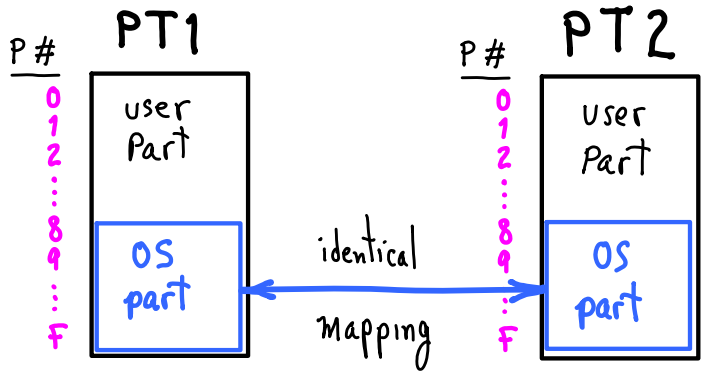


NOTE

--- OS can move OS pages around, or to disk.
Just change PT.

--- Multiple processes?
multiple PTs;
PTs in OS data area;
switch PTBR to point to current PT.
OS part of both PTs is the same.



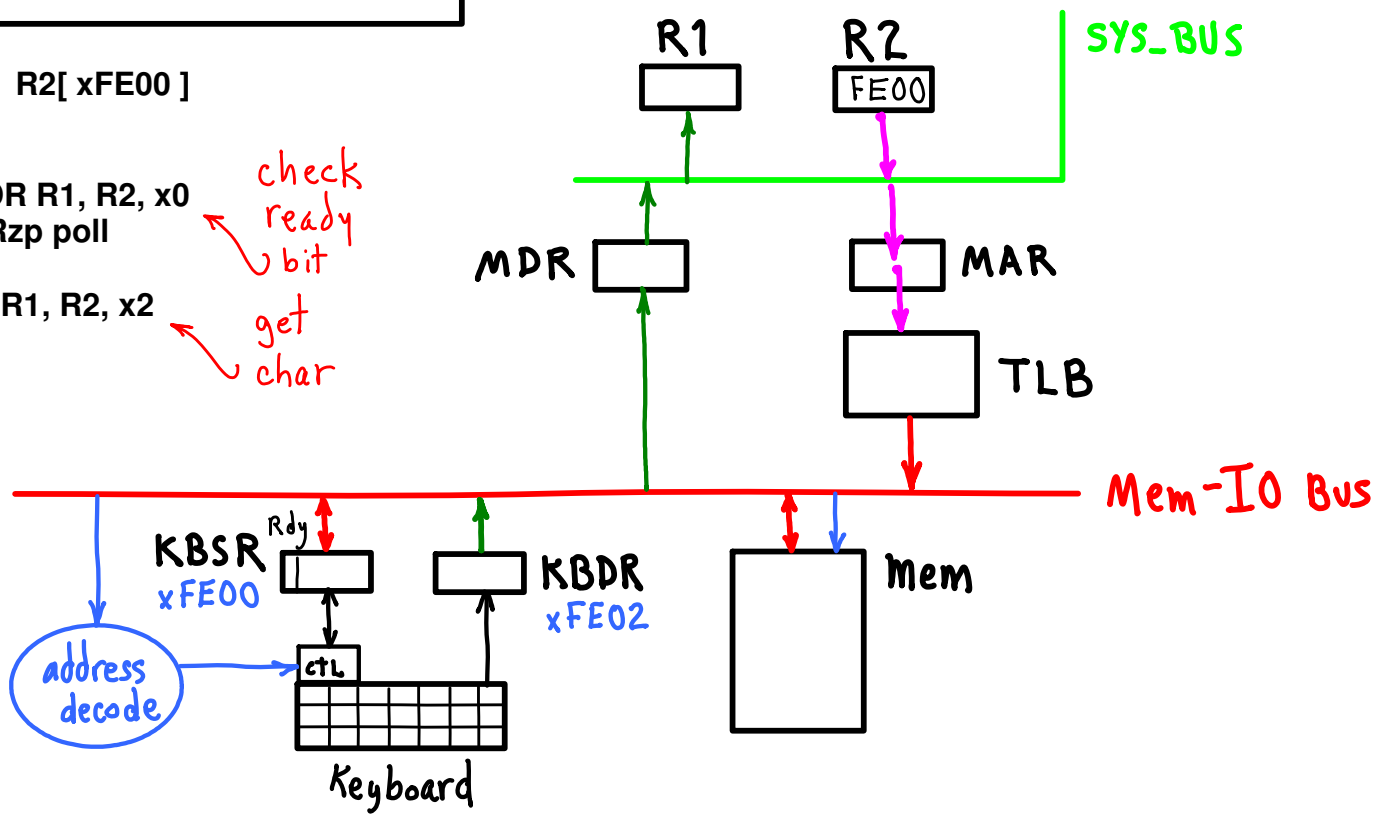


VMem, cache, I/O

R2[xFE00]

poll:
LDR R1, R2, x0
BRzp poll

LDR R1, R2, x2

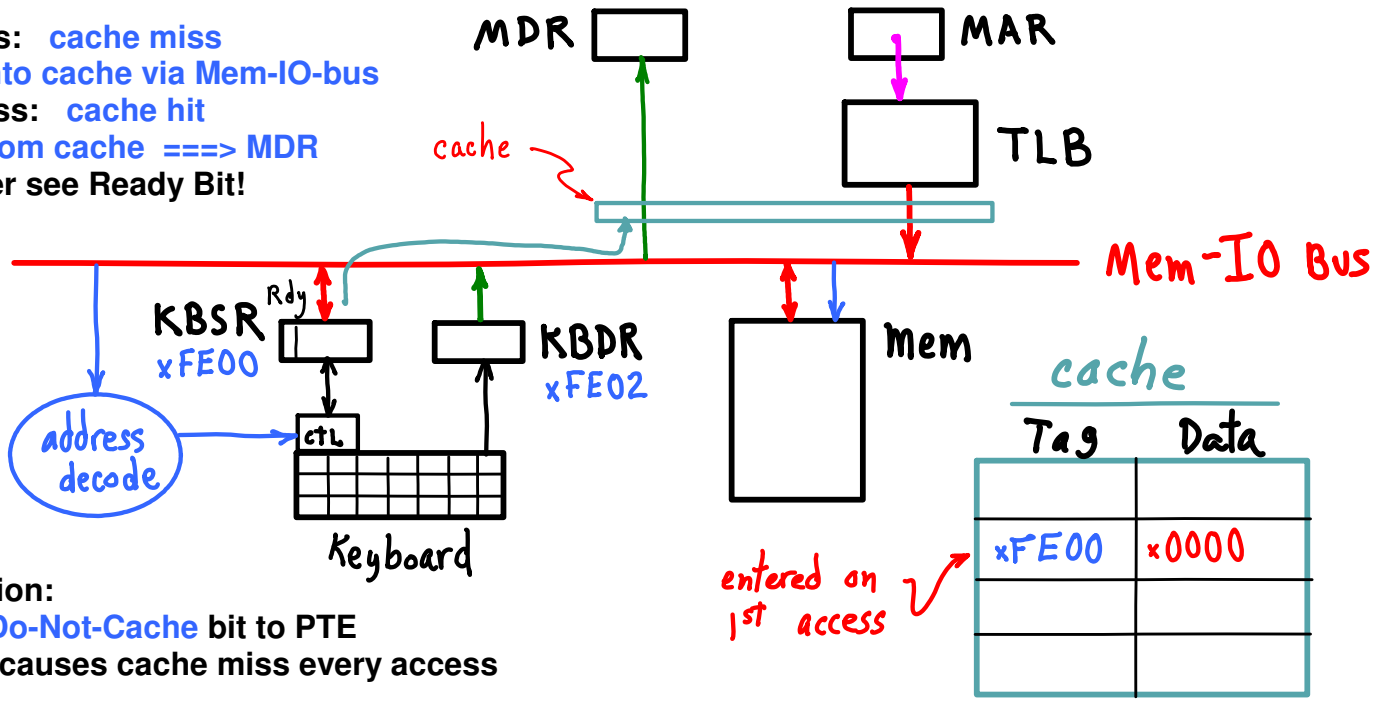


Device address decode recognizes xFE00 and xFE02.
Keyboard data moved to R1.

Suppose TLB[xF x4] xFE00 ===> x4E00 (references a word in memory!)

Solution: TLB[xF xF] xFE00 ===> xFE00 (accesses KBSR)

1st access: cache miss
x0000 into cache via Mem-IO-bus
N-th access: cache hit
x0000 from cache ===> MDR
===> never see Ready Bit!



Solution:
Add Do-Not-Cache bit to PTE
===> causes cache miss every access