Memory Protection

Lecture 9

Memory Protection

- Memory Protection Unit
- Memory Management Unit

Memory Protection

ARM: MPU, RISC-V: PMP

Bibliography

for this section

Joseph Yiu, *The Definitive Guide to ARM® Cortex®-M0 and Cortex-M0+ Processors, 2nd Edition*

- Chapter 12 - *Memory Protection Unit*

Memory Protection

memory access defined region by region

- **restricts** access to **physical memory**
- uses **physical addresses**

The processor works in three modes:

- **machine** mode (*optional*) - used at boot, allows access to everything
- **supervisor** mode - restricts access to some registers and accesses memory through Memory Protection EL2 (*if machine mode exists*)
- **user** mode - allows only ALU and memory access through Memory Protection
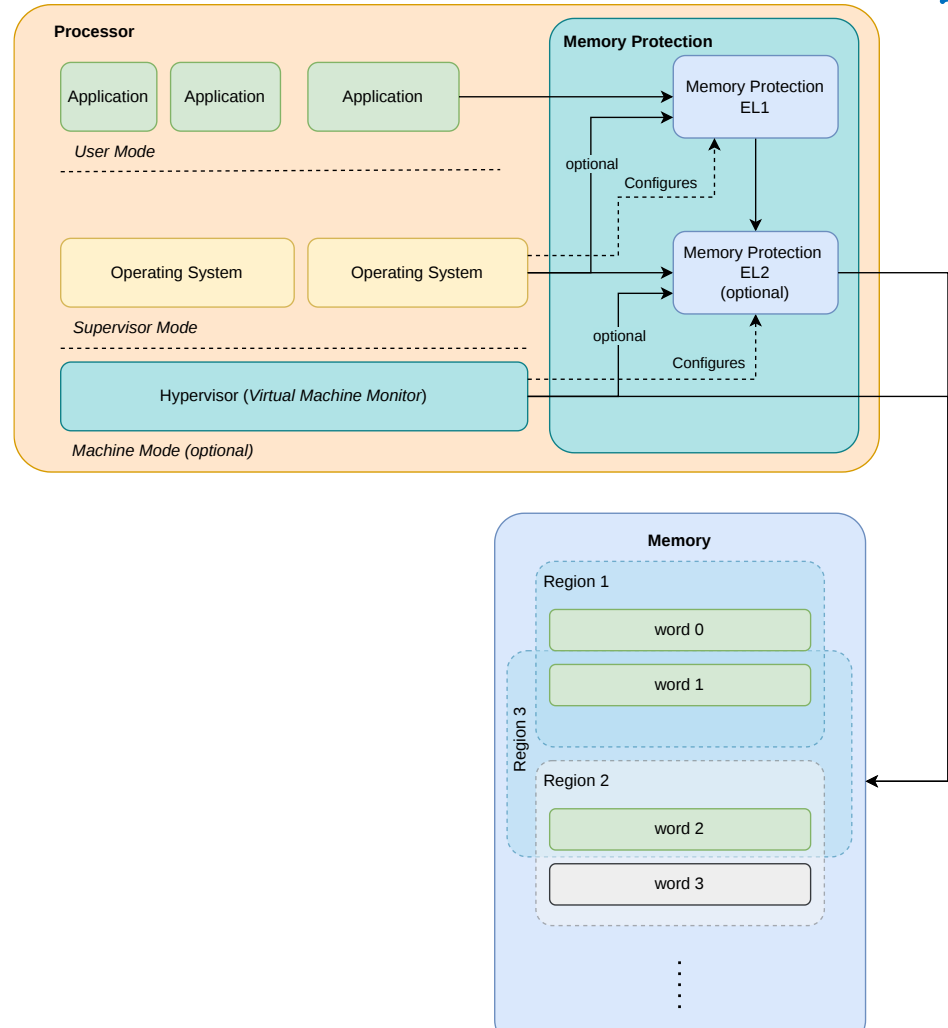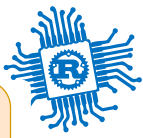
MPU for RP2040

Cortex-M0+ works in three modes

- **handler** mode - *no restrictions* - used while executing ISRs and Exception Handlers
- **thread** mode
  - **privileged** *no restrictions* - usually used for the operating system
  - **unprivileged** mode - *allows only ALU and memory access through Memory Protection* - used for applications
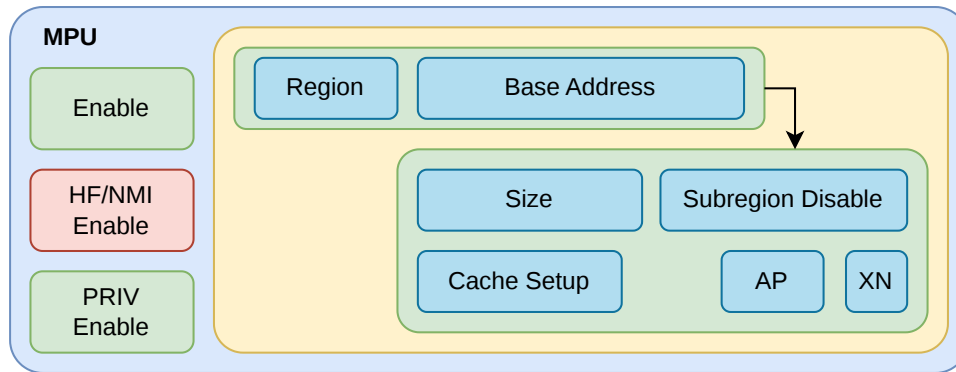
MPU allows 8 regions

- each region has up to 8 subregions
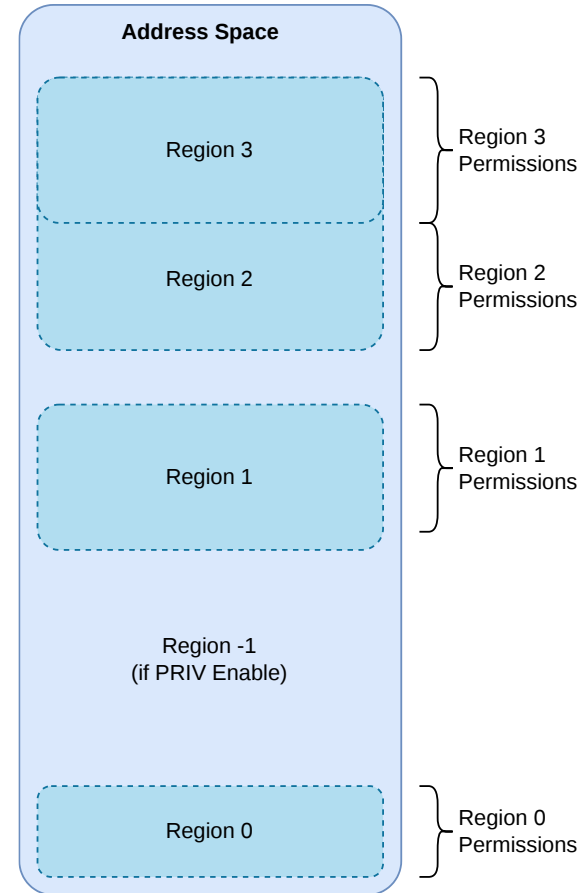- permissions R W X
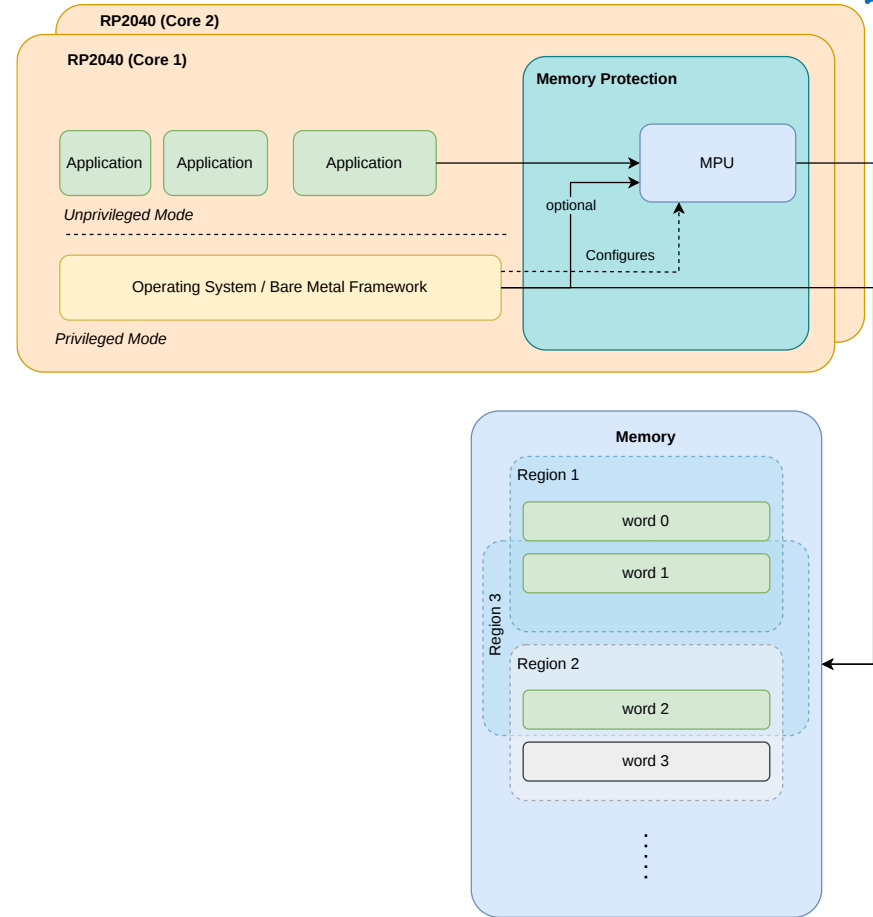
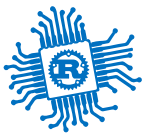# Memory Protection Unit

Cortex-M MPU

- allows the definition of *memory regions*

- regions can overlap, *highest region number* takes *priority*

- regions have access permissions (similar to rwx)

$$region\_size = min(256, 2^{size})$$

$$base\_address = region\_size \times N$$

Memory Protection Unit

Access Protection
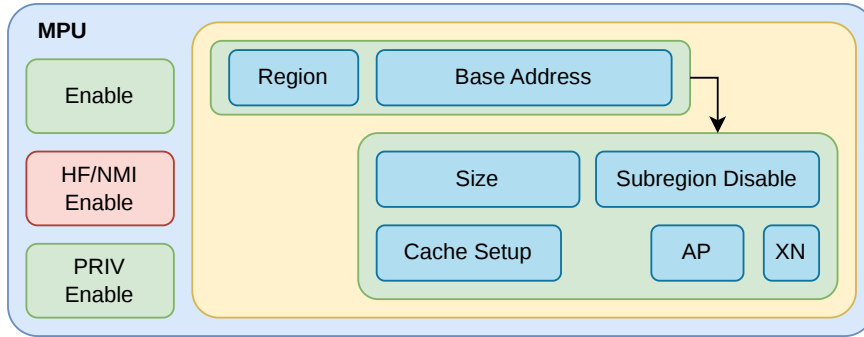
AP Access Protection

XN eXecute Never

- faults if MCU has to read the next instruction from an *XN* region

AP	Privileged Mode	Unprivileged Mode
000	No Access	No Access
001	Read/Write	No Access
010	Read/Write	Read only
011	Read/Write	Read/Write
100	Do not use	Do not use
101	Read only	No Access
110	Read only	Read only
111	Read/Write	Read only

Subregions

- each region is divided in 8 subregion
- each bit in `Subregion Disable` disables a subregion
- a disabled subregion triggers a fault if accessed
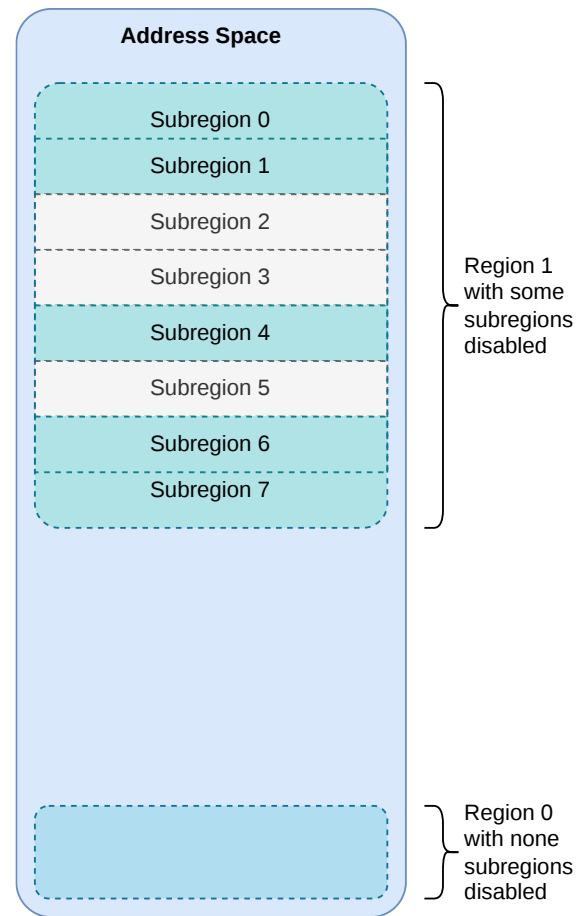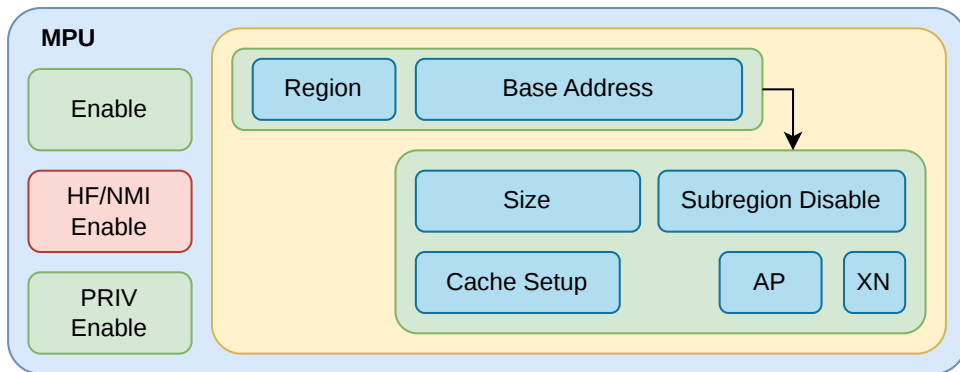
Subregions' Usage

improve granularity

$$region\_size = min(256, 2^{size})$$

$$base\_address = region\_size \times N$$

$$subregion\_size = \frac{region\_size}{8}$$

- a 5K region is not allowed (5K is not a power of 2)
- use two 4K regions back to back
- disable 6 of the subregions (subregion is 512B)

Address Space

. . . .

Data (5K)

Region 1 with 6 subregions disabled

Region 0 no subregion disabled

. . . .
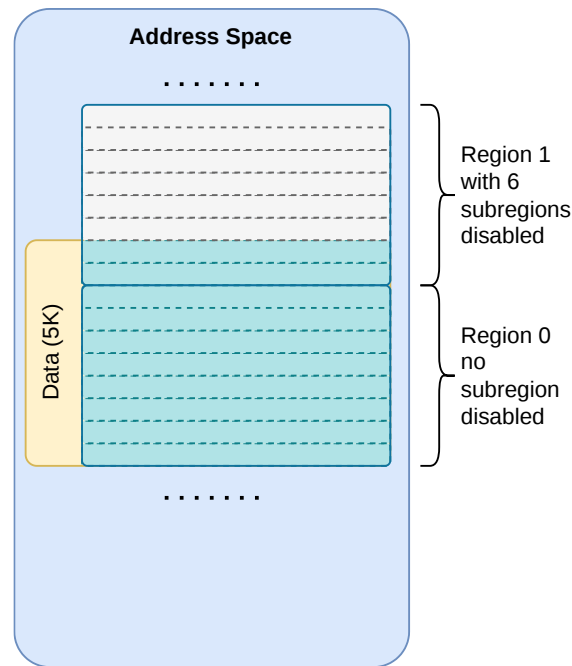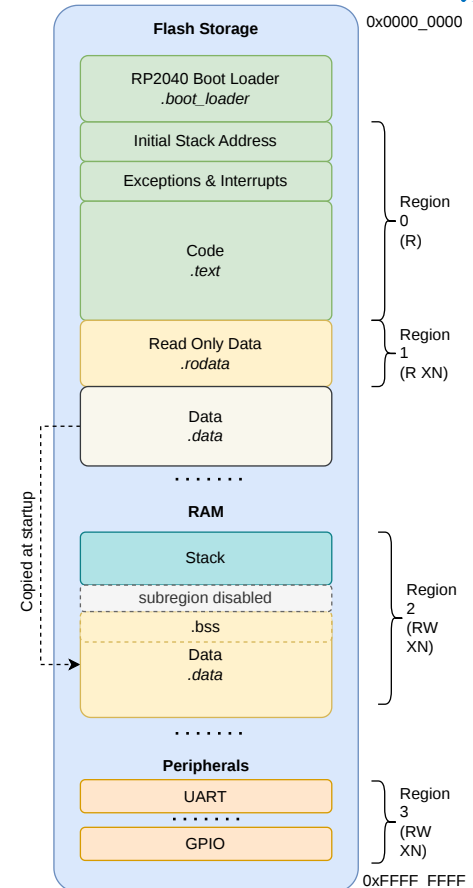
Memory Layout

protection

Flash

- **Code** - *read* and *execute*

- **.rodata** - constants - *read only*

- **.data** - *in flash* - initialized global variables

  - is copied to RAM at startup by the `init` function

  - *should not be accessed after startup*

RAM

- **stack** - *read* and *write*

  - *usually protected by unaccessible memory before and after*

- **.data** - *in RAM* - global variables - *read* and write

- **.bss** - global variables (not initialized or initialized to `0`) - *read* and *write*

Flash Storage

0x0000_0000

| RP2040 Boot Loader .boot_loader |
| Initial Stack Address |
| Exceptions & Interrupts |
| Code .text |

Region 0 (R)

| Read Only Data .rodata |

Region 1 (R XN)

| Data .data |
. . . .

RAM

| Stack |
| subregion disabled |
| .bss |
| Data .data |

Region 2 (RW XN)

. . . .

Peripherals

| UART |
. . . .
| GPIO |

Region 3 (RW XN)

0xFFFF_FFFF

Copied at startup

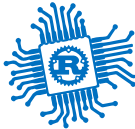* drawing is not at scale, code and data are significantly greater than the interrupt vector

Memory Management

MMU

Bibliography

for this section

1. **Andrew Tanenbaum**, *Modern Operating Systems (4th edition)*

   - Chapter 3 - *Memory Management*
     - Subchapter 3.3 - *Virtual Memory*

2. **Philipp Oppermann**, *Writing an OS in Rust*

   - *Introduction to Paging*
   - *Paging Implementation*
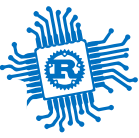
Memory Management

memory access defined page by page

- uses *logical addresses*
- **translates** to *physical addresses*

The processor works in at least two modes:

- **supervisor** mode
  - restricts access to some registers
  - accesses virtual addresses through Memory Protection (*if machine mode exists*)
- **user** mode
  - allows only ALU and memory load and store
  - accesses memory access through the Memory Management Unit (*MMU*)
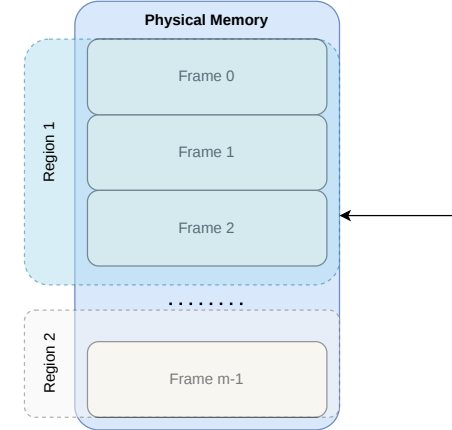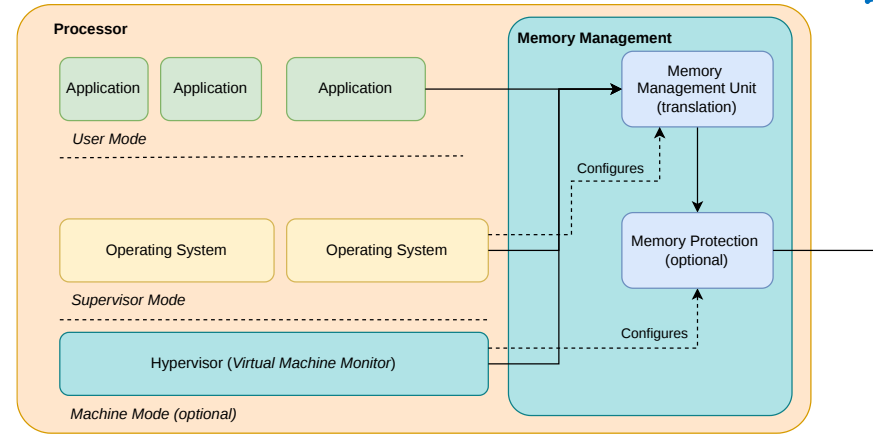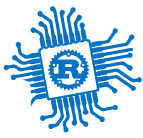
Paging

the memory unit *is the page*

- Physical Memory (*RAM*) is divided in **frames**
- Logical Memory is divided in **pages**
- *page = frame =* **4 KB** (usually)

logical addresses are translated to *physical addresses* using a **page table**

the **page table** is located in the **physical memory**

- each memory access requires at least memory 2 accesses

Logical Memory

Page 0
Page 1
Page 2
Page 3
Page 5
. . . .
Page n-1

Page Table

1
0
2
7
N/A
. . . .
200

Physical Memory

Frame 0
Frame 1
Frame 2
. . . .
Frame m-1
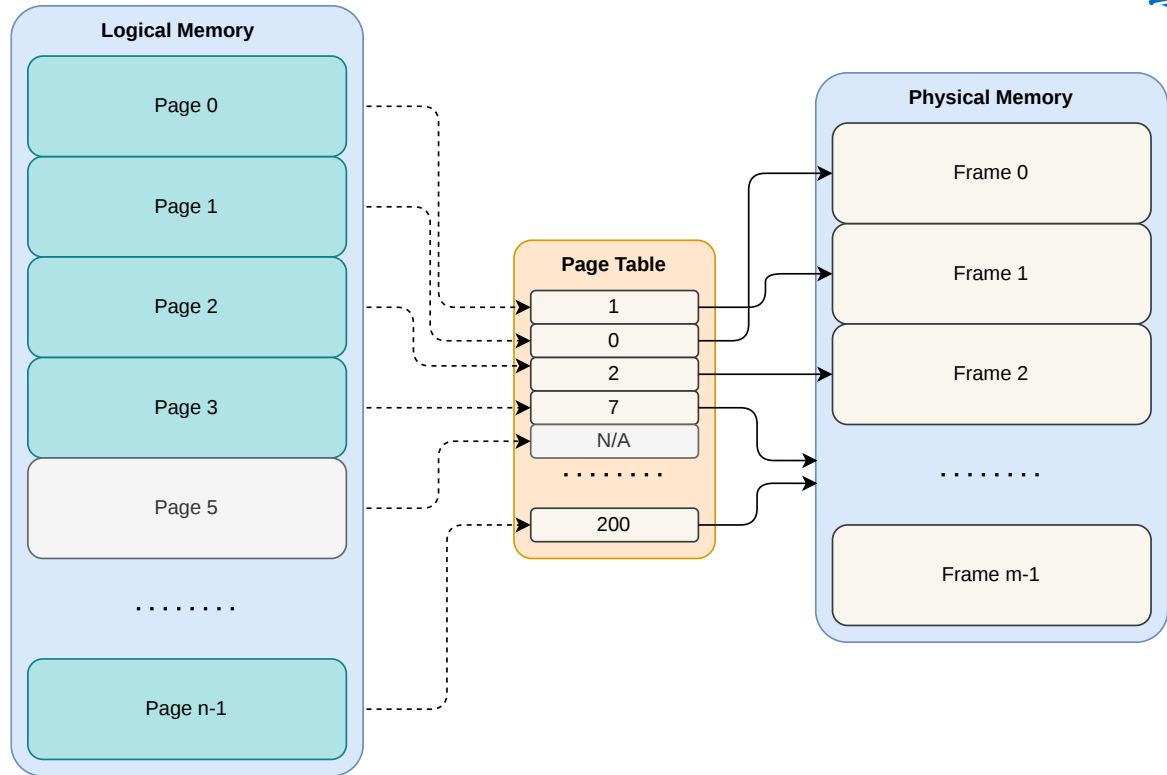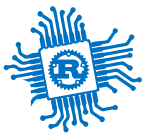
Address Translation

page to frame

the logic address is divided in two parts:

- *page index*
- *offset* within the page

the MMU translates every logic address into a physical address using a *page table*
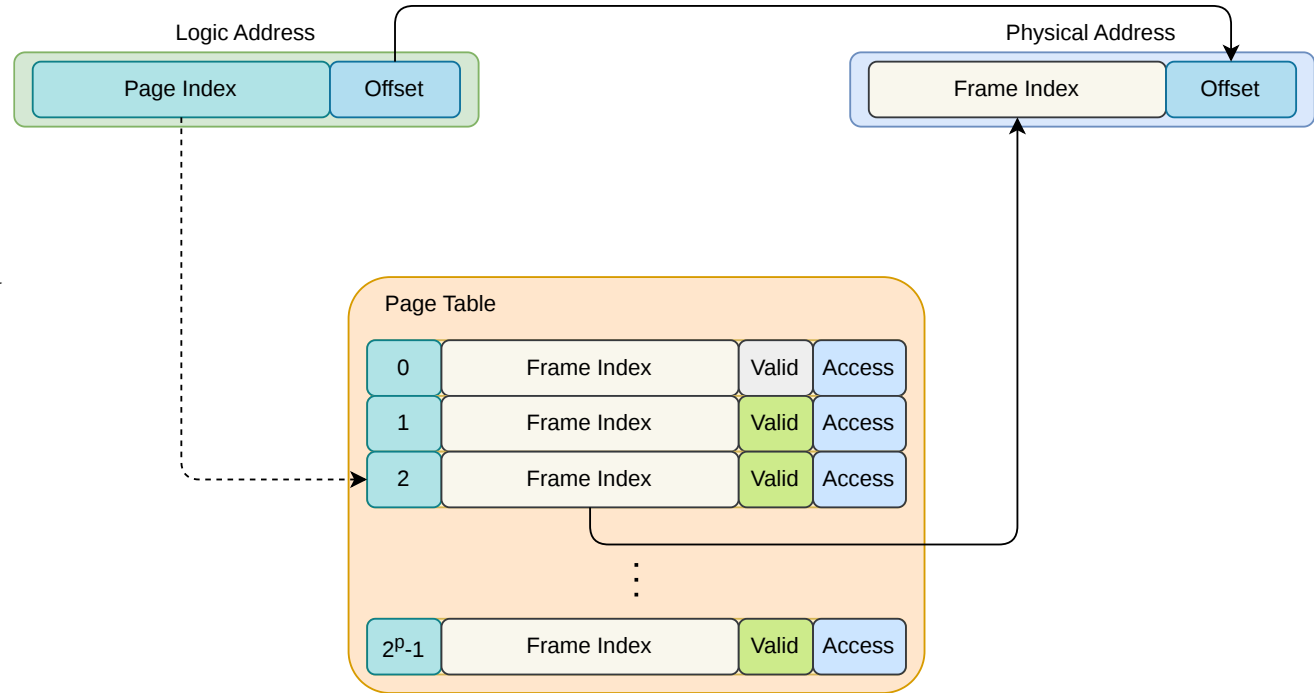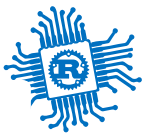
Translation Lookaside Buffer (TLB)

caching address translation

the **page table** is **stored in RAM**

each memory access **requires 2 accesses**

1. read the page table entry to translate the address
2. the requested access

Logic Address

| Page Index | Offset |

Physical Address

| Frame Index | Offset |

Translation Lookaside Buffer

TLB Hit

TLB MIss

Page Table

0	Frame Index	Valid	Access
1	Frame Index	Valid	Access
2	Frame Index	Valid	Access
⋮			
2^p-1	Frame Index	Valid	Access
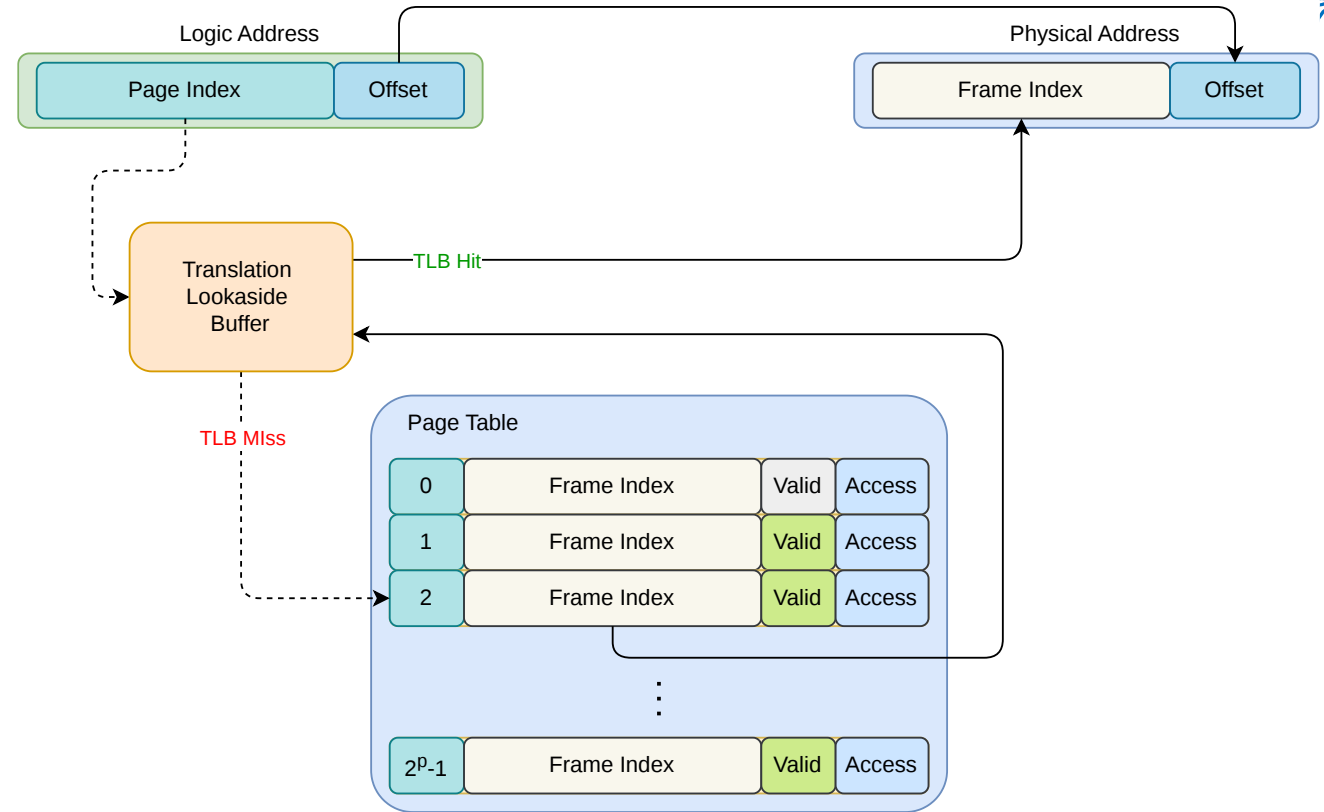
Page Directory

caching address translation

$$size_{table} = \frac{size_{ram}}{size_{page}}$$

- each table entry is 4B
- the address space is 4GB (for 32 bits processors)

$$size_{table\_32\_bits} = \frac{2^{32}}{4 \times 2^{10}}$$

$$size_{table\_32\_bits} = 4MB$$

RAM was counted in MB when paging started being used

Logic Address

Page Directory	Page Index	Offset

Physical Address

Frame Index	Offset

Translation Lookaside Buffer — TLB Hit

TLB Miss

Page Directory

0	Page Table Frame	Valid	Access
1	Page Table Frame	Valid	Access
2	Page Table Frame	Valid	Access
⋮			
2^p-1	Page Table Frame	Valid	Access

Page Table

0	Frame Index	Valid	Access
1	Frame Index	Valid	Access
2	Frame Index	Valid	Access
⋮			
2^p-1	Frame Index	Valid	Access

two levels, page directory and table, usually used for 32 bits systems

Page Table Entry

for x86 - 32 bits

this is one entry of the page table

- **P** - is the page's frame present in RAM?
- **R/W** - read only or read write access
- **U/S** - can the page be accessed in user mode?
- **D** and **A** - has this page been written since the OS has reset these bits?
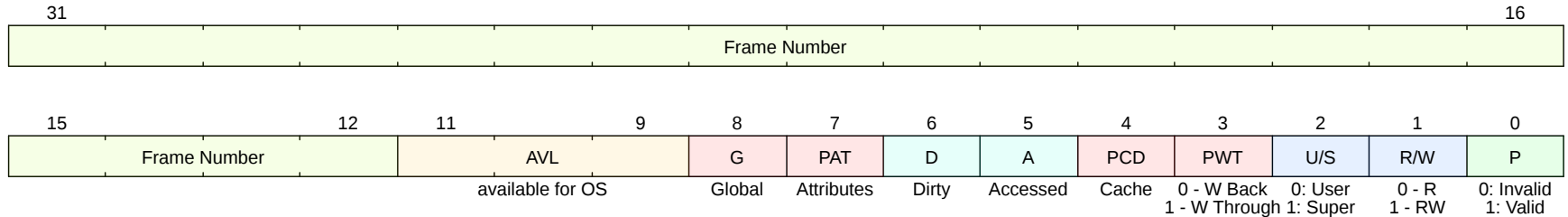- **AVL** - bits available for the OS to use, ignored by MMU

31															16
Frame Number															

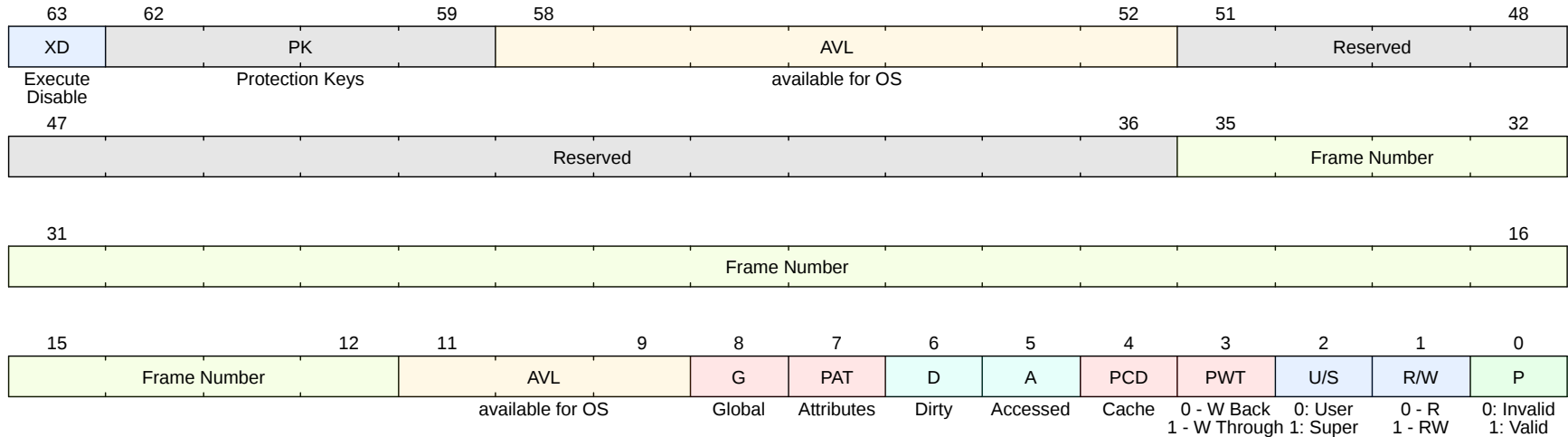| 15	12	11	9	8	7	6	5	4	3	2	1	0
Frame Number		AVL		G	PAT	D	A	PCD	PWT	U/S	R/W	P
		available for OS		Global	Attributes	Dirty	Accessed	Cache	0 - W Back / 1 - W Through	0: User / 1: Super	0 - R / 1 - RW	0: Invalid / 1: Valid

Page Table Entry

for x86 - 32 bits with PAE

this is one entry of the page table using Physical Address Extension (*PAE*)

- **XD** - eXecute Disable (aka *DEP*), if set triggers a fault if an instruction is read from the page

- **PK** - Protection Keys, allows user mode to set protection (64 bit only)

63	62	59	58	52	51	48
XD	PK		AVL		Reserved	
Execute Disable	Protection Keys		available for OS			

47	36	35	32
Reserved		Frame Number	

31	16
Frame Number	

| 15	12	11	9	8	7	6	5	4	3	2	1	0
Frame Number		AVL		G	PAT	D	A	PCD	PWT	U/S	R/W	P
		available for OS		Global	Attributes	Dirty	Accessed	Cache	0 - W Back 1 - W Through	0: User 1: Super	0 - R 1 - RW	0: Invalid 1: Valid

Microcontroller (MCU)

Integrated in embedded systems for certain tasks

- low operating frequency (MHz)
- a lot of I/O ports
- controls hardware
- does not require an Operating System
- costs $0.1 - $25
- uses **Memory Protection Unit**
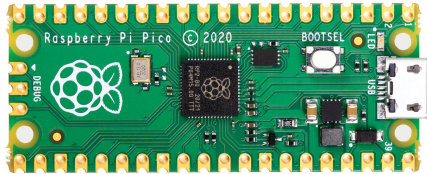
Microprocessor (CPU)

General purpose, for PC & workstations

- high operating frequency (GHz)
- limited number of I/O ports
- usually requires an Operating System
- costs $75 - $500
- uses **Memory Management Unit**
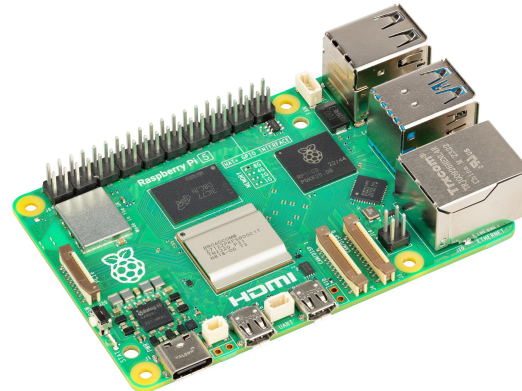
Conclusion

we talked about

- Memory Protection Unit
- Memory Management Unit