Zadanie 1 ::

```
0x027c = 0000001001111100
0x03a9 = 0000001110101001
0x0040 = 0000000001000000
```

Zadanie 2 ::

```
Page size = 4KiB, so we needs 12 bits to save page offset
And then we have:
19-12 -- VPN
11-00 -- VPO


_________________________________
0x04669 = 00000100 011001101001

VPN = 0x04
VPO = 0x669

TLBI = 0x00
TLBT = 0x01

TLB MISS
PPN = 0x09

UPDATE TLB WITH NEW PPN
NEW TABLE

__________________
| 1 | 11 | 1 | 12 |
| 1 |  7 | 2 |  4 |
| 1 |  3 | 3 |  6 |
| 1 |  4 | 0 |  9 |

_________________________________
0x02227 = 00000010 001000100111

VPN = 0x02
VPO = 0x227

TLBI = 0x02
TLBT = 0x00

TLB MISS

____________________
| 1 | 11 | 2 | 12 |
| 1 |  7 | 3 |  4 |
| 1 |  0 | 0 | 13 |
| 1 |  1 | 1 |  9 |

_________________________________
0x13916 = 00010011 100100010110

VPN = 0x13
VPO = 0x916

TLBI = 0x3
TLBT = 0x4

TLB HIT


____________________
| 1 | 11 | 2 | 12 |
```

```
| 1 |  7 | 3 |  4 |
| 1 |  0 | 1 |  d |
| 1 |  1 | 0 |  9 |

--------------------------------
0x34587 = 00110100 010110000111

VPN = 0x34
VPO = 0x587

TLBI = 0x00
TLBT = 0x0d

TLB MISS
PAGE FAULT

| 13 | 1 | ??? |


--------------------
| 1 | 11 | 3 | 12 |
| 1 | 13 | 0 | ?? |
| 1 |  0 | 2 | 13 |
| 1 |  1 | 1 |  9 |

--------------------------------
0x48870 = 01001000 100001110000

VPN = 0x48
VPO = 0x870

--------------------------------
0x12608 = 00010010 011000001000

VPN = 0x12
VPO = 0x608

--------------------------------
0x49225 = 01001001 001000100101

VPN = 0x49
VPO = 0x225
```

Zadanie 3 ::

> 4 KiB is widely popuplar page granularity in other architectures too. One could argue that this size comes from the division of a 32-bit virutal address into two 10-bit indexes in page directories/tables and the remaining 12 bits give the 4 KiB page size.

EAT – Effective Access Time

p – page fault rate ($0 \leq p \leq 1$)

MA – Memory Access

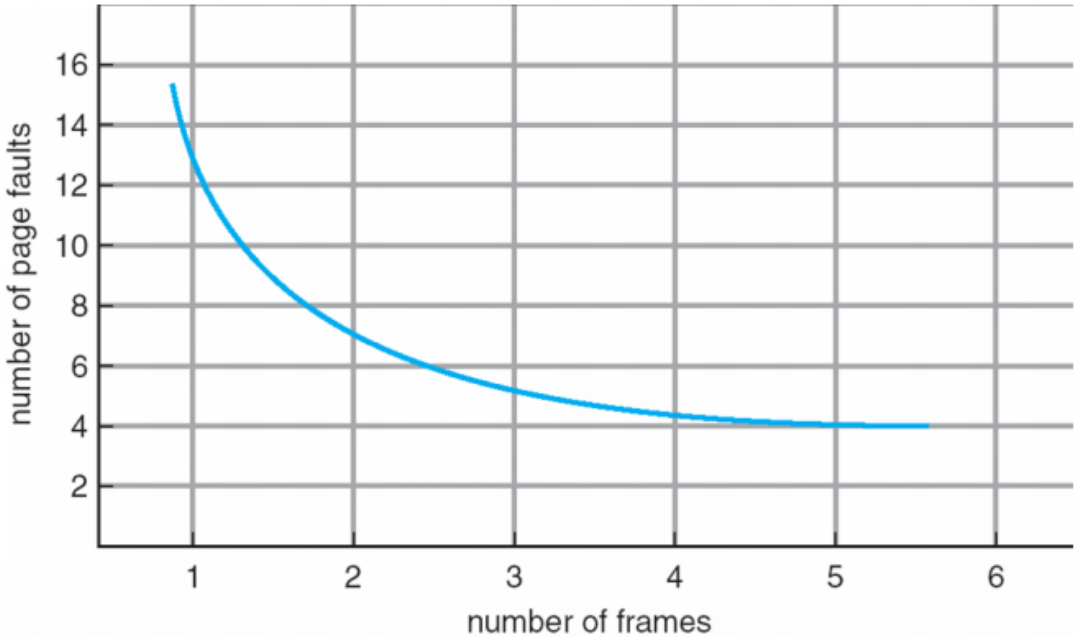
PFO – Page Fault Overhead

SPO – Swap Page Out

SPI – Swap Page In

RO – Restart Overhead

$$EAT = (1 - p) * MA + p * (PFO + SPO + SPI + RO)$$



- **If (working set size < main memory size)**
  - Good performance for one process after compulsory misses

- **If (SUM(working set sizes) > main memory size)**
  - *Thrashing:* Performance meltdown where pages are swapped (copied) in and out continuously

Zadanie 4 ::

```
Address len. [AL]: 32 bits
Page size    [PS]: 4KiB = 4096B
Entry size   [ES]: 4B
Used memory  [UM]: 1GiB = 1024MiB = 1048576KiB

a)
    ~Rozmiar tablicy to po prostu (UM / PS) * ES = 1048576KB = 1024GB = 1TB~
    Rozmiar tablicy to 2^32/2^12 * 4 = 4MB

b)
    Wiemy, że katalog stron ma 1024 wpisy.
    Możemy zatem podzielić adres na części: 10-10-12, gdzie pierwsza część adresuje tablice
        pierwszego poziomu, a drugie dziesięć - drugiego poziomu.

    Pierwsza tablica będzie zawsze ważyć 2^10 * 4 = 4KB
    Nam wystarczy 256 wpisów a reszta może pozostać pusta
    Czyli w najlepszym przypadku zajmie nam to 256*4K + 4K = 1.003MB
    W najgorszym, zakładając że dane są rozrzucone po całej pamięci
    1025 * 4KB = 4.0039MB
```

Zadanie 5 ::

> On an address space switch, as occurs on a process switch but not on a thread switch, some TLB entries can become invalid, since the virtual-to-physical mapping is different. The simplest strategy to deal with this is to completely flush the TLB. This means that after a switch, the TLB is empty and any memory reference will be a miss,

and it will be some time before things are running back at full speed. Newer CPUs use more effective strategies marking which process an entry is for. This means that if a second process runs for only a short time and jumps back to a first process, it may still have valid entries, saving the time to reload them.

Pomysłem na to jest próba zapobiegnięcia musu flushowania całego TLB tagując odpowiednie adresy dla odpowiednich procesów. (np. po PID'ie)

Zadanie 7 ::

Both tables contain 1024 4-byte entries, making them 4 KiB each. In the page directory, each entry points to a page table. In the page table, each entry points to a physical address that is then mapped to the virtual address found by calculating the offset within the directory and the offset within the table. This can be done as the entire table system represents a linear 4-GiB virtual memory map.

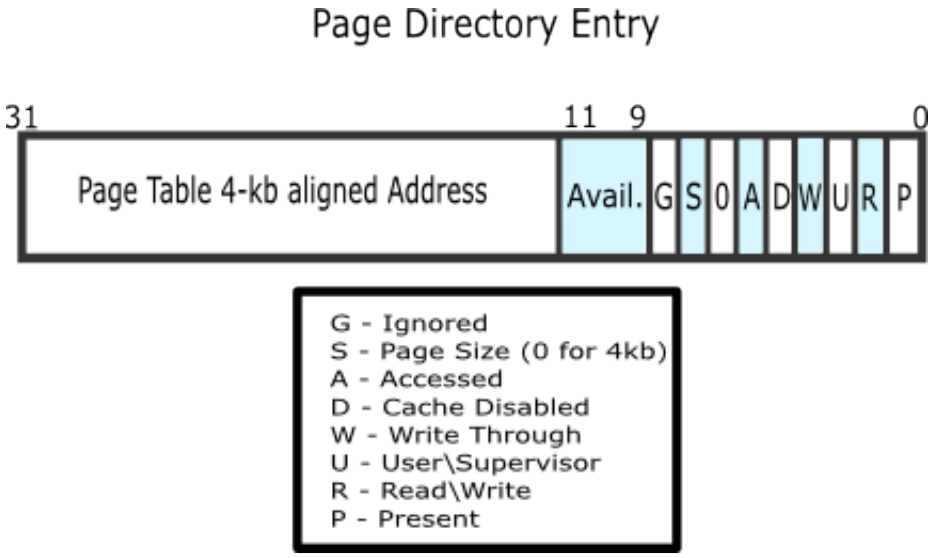


Figure 1: PDE

S, or 'Page Size' stores the page size for that specific entry. If the bit is set, then pages are 4 MiB in size. Otherwise, they are 4 KiB. Please note that 4-MiB pages require PSE to be enabled.

A, or 'Accessed' is used to discover whether a page has been read or written to. If it has, then the bit is set, otherwise, it is not. Note that, this bit will not be cleared by the CPU, so that burden falls on the OS (if it needs this bit at all).

D, is the 'Cache Disable' bit. If the bit is set, the page will not be cached. Otherwise, it will be.

W, the controls 'Write-Through' abilities of the page. If the bit is set, write-through caching is enabled. If not, then write-back is enabled instead.

U, the 'User/Supervisor' bit, controls access to the page based on privilege level. If the bit is set, then the page may be accessed by all; if the bit is not set, however, only the supervisor can access it. For a page directory entry, the user bit controls access to all the pages referenced by the page directory entry. Therefore if you wish to make a page a user page, you must set the user bit in the relevant page directory entry as well as the page table entry.

R, the 'Read/Write' permissions flag. If the bit is set, the page is read/write. Otherwise when it is not set, the page is read-only. The WP bit in CR0 determines if this is only applied to userland, always giving the kernel write

access (the default) or both userland and the kernel (see Intel Manuals 3A 2-20).

P, or 'Present'. If the bit is set, the page is actually in physical memory at the moment. For example, when a page is swapped out, it is not in physical memory and therefore not 'Present'. If a page is called, but not present, a page fault will occur, and the OS should handle it. (See below.)

> The remaining bits 9 through 11 are not used by the processor, and are free for the OS to store some of its own accounting information. In addition, when P is not set, the processor ignores the rest of the entry and you can use all remaining 31 bits for extra information, like recording where the page has ended up in swap space.
>
> Setting the S bit makes the page directory entry point directly to a 4-MiB page. There is no paging table involved in the address translation. Note: With 4-MiB pages, bits 21 through 12 are reserved! Thus, the physical address must also be 4-MiB-aligned.

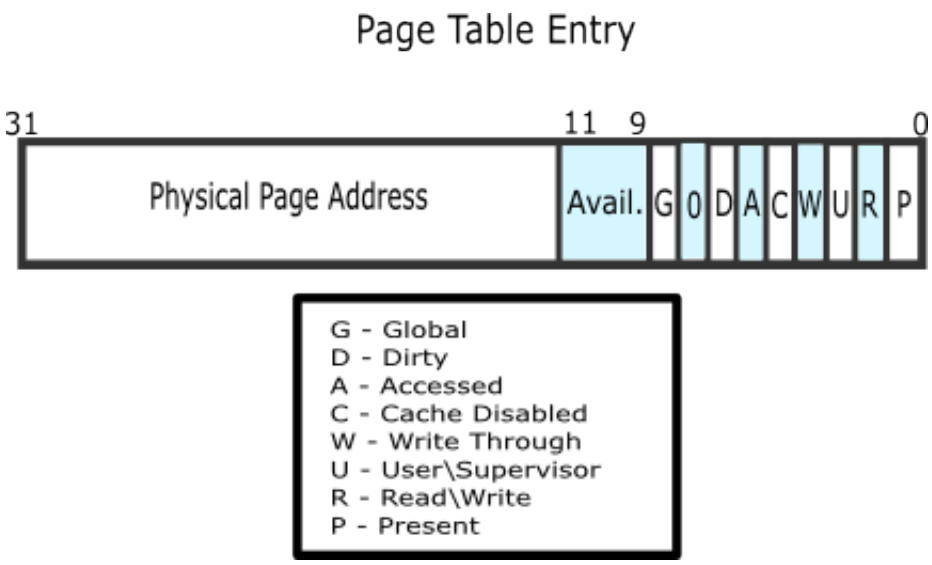


Figure 2: PTE

The first item, is once again, a 4-KiB aligned physical address. Unlike previously, however, the address is not that of a page table, but instead a 4 KiB block of physical memory that is then mapped to that location in the page table and directory.

The Global, or 'G' above, flag, if set, prevents the TLB from updating the address in its cache if CR3 is reset. Note, that the page global enable bit in CR4 must be set to enable this feature.

If the Dirty flag ('D') is set, then the page has been written to. This flag is not updated by the CPU, and once set will not unset itself.

The 'C' bit is 'D' bit above.

Zadanie 9 ::

> The inverted page table (IPT) is best thought of as an off-chip extension of the TLB which uses normal system RAM. Unlike a true page table, it is not necessarily able to hold all current mappings. The OS must be prepared to handle misses, just as it would with a MIPS-style software-filled TLB.
>
> The IPT combines a page table and a frame table into one data structure. At its core is a fixed-size table with the number of rows

equal to the number of frames in memory. If there are 4000 frames, the inverted page table has 4000 rows. For each row there is an entry for the virtual page number (VPN), the physical page number (not the physical address), some other data and a means for creating a collision chain, as we will see later.

To search through all entries of the core IPT structure is inefficient, and a hash table may be used to map virtual addresses (and address space/PID information if need be) to an index in the IPT - this is where the collision chain is used. This hash table is known as a hash anchor table. The hashing function is not generally optimized for coverage - raw speed is more desirable. Of course, hash tables experience collisions. Due to this chosen hashing function, we may experience a lot of collisions in usage, so for each entry in the table the VPN is provided to check if it is the searched entry or a collision.

In searching for a mapping, the hash anchor table is used. If no entry exists, a page fault occurs. Otherwise, the entry is found. Depending on the architecture, the entry may be placed in the TLB again and the memory reference is restarted, or the collision chain may be followed until it has been exhausted and a page fault occurs.

A virtual address in this schema could be split into two, the first half being a virtual page number and the second half being the offset in that page.

A major problem with this design is poor cache locality caused by the hash function. Tree-based designs avoid this by placing the page table entries for adjacent pages in adjacent locations, but an inverted page table destroys spatial locality of reference by scattering entries all over. An operating system may minimize the size of the hash table to reduce this problem, with the trade-off being an increased miss rate. There is normally one hash table, contiguous in physical memory, shared by all processes. Memory fragmentation makes per-process page tables impractical, so a per-process identifier is used to disambiguate the pages of different processes from each other. It is somewhat slow to remove the page table entries of a process; the OS may avoid reusing per-process identifier values to delay facing this or it may elect to suffer the huge waste of memory associated with pre-allocated (necessary because of fragmentation) per-process hash tables.

Program memory references are 32-bit logical addresses. The 4 high order bits of the logical address index a set of segment registers, each of which contains a 24-bit `virtual segment identifier''(VSID). The logical address is concatenated with the VSID to produce a virtual address. There is a translation look-aside buffer of cached virtual $\rightarrow$ physical translations and hashed page tables indexed by a (hashed) virtual address. The tables are organized into`buckets", each consisting of eight page table entries (PTEs). Each PTE contains a 20-bit physical page address, a 24-bit virtual segment identifier (VSID) and permission and other housekeeping information. Once a TLB miss occurs, a hash function is computed on the virtual address to obtain the index of a bucket. If no matching entry is found in this bucket, a secondary hash function is computed to find the index of an overflow bucket. If no entry is found in either bucket, the OS must determine further action. On the 604, a TLB miss causes the hardware to compute the hash function and search the hash table. If no match is found, a page fault interrupt is generated and a software handler is started. On the

603, there are registers to assist hashing even though the hardware does not require software to store PTEs in a hash table. Since a TLB miss is handled in hardware, the 604 has a hash-table miss interrupt rather than a TLB miss interrupt.

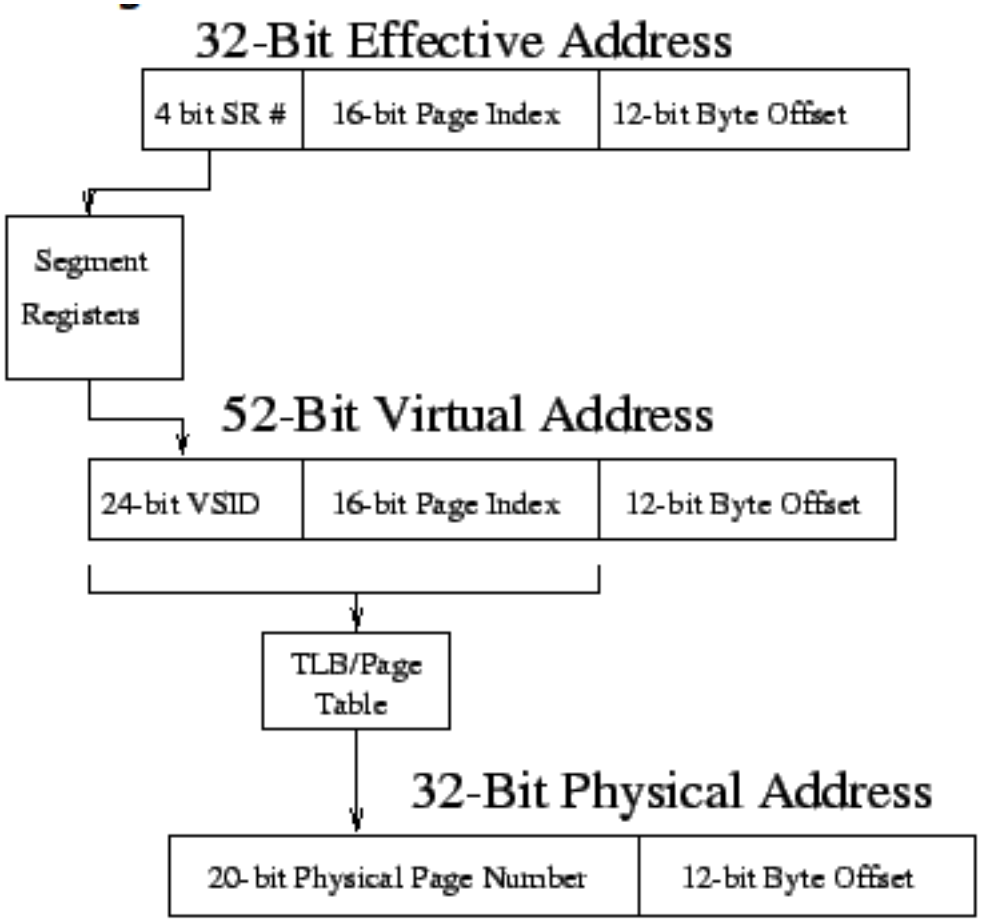


Figure 3: How this works