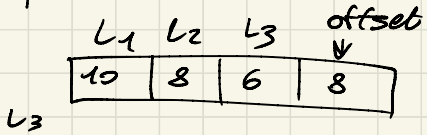
1) 32 bit machine, addresses divided in parts of 10, 8, 6 and 8 bits
3 level page table system
single entry is 32 bits. Byte addressed memory



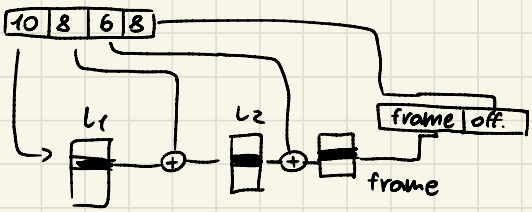
L3

1) what is page size of the system?

2^8 because we have an 8 bit offset

2) suppose a process requires 256K of memory (which starts at address 0)
how much space does the page tables of this process take?

- 1 page 2^8 bytes
- 1 L3 table 2^6 pages entry
- 1 L2 table 2^8 L3 tables



- 256KB = 2^{18} B
  ⇓ $2^{18}/2^8$
  2^{10} pages
  ⇓ for this I need 2^4 L3 tables
  for this 1 L2 and 1 L1 tables

$$4 \cdot (2^4 \cdot 2^6 + 1 \cdot 2^8 + 1 \cdot 2^{10}) = 9216\ B$$

3) code segment 48K from virt. address X10000000, 600K data segment from address 0X80000000, stack segment of 64K from 0XF000 0000
How much space page table require?

$2^6 = 64 \cdot 3 = 192$

- 48 KB → 192 pages → 3 L3 tables, 1 L2, 1 L1

  $48 \cdot 2^{10}/2^8 = 48 \cdot 2^2 = 48 \cdot 4 = 192$
- 600KB 2400 pages → 38 L3 tables, 1 L2, 1 L1

  $600 \cdot 2^{10}/2^8 = 600 \cdot 4 = 2400$
- 64KB → 256 pages → 4 L3 tables, 1 L2, 1 L1

  $2^6 \cdot 2^{10}/2^8 = 2^8$

check if they are under different L_4

L_3 bits ↓ off ↓

1 L_3 table "covers" $2^6 \cdot 2^8 = 2^{14}$ B → 0x4 0000

1 L_2 table "covers" 2^8 L_3 tables → 2^{22} B → 0x 400 0000

so big it contains all L_2 tables

1 L_1 table "covers" 2^{10} L_2 tables → 2^{32} B → 0x1 0000 0000

$4 \cdot \left((3+38+4) \cdot 2^6 + 3 \cdot 2^8 + 1 \cdot 2^{10} \right)$ ← only one L_4 table for all

$= 18688$ B

2) Determine the number of levels in a multilevel paging system

virtual addresses of 36 bits, pages 8K size.

Page entry 8 Bytes long

1) How many pages does the virt. address space has?

$$\frac{2^{36}}{8 \cdot 2^{10}} = \frac{2^{36}}{2^{13}} = 2^{23} \text{ pages}$$

2)

average process size of 8GB. How many levels are most suitable if we work with a multilevel page table (1 to 3)?

Weight pros and cons.

For the 2-level system choose the size of a level 2 page table so that it fits exactly into a single page.

for 3 levels take a more or less equal division for each of the fields.

- single level

L1	offset
23	13

- 2^{23} page table entries $\cdot$ 8 $\frac{\text{byte}}{\text{PTE}}$ = 2^{26} Byte = 64MB

- 2 level

L1	L2	offset
13	10	13

- 1 page = 2^{13} B (contains 2^{10} page table entries)
- 1 L2 table = 2^{10} pages
- 8GB = 2^{33} B so 2^{20} pages, 2^{10} L2 pages and 1 L1 page

↑ every page is 2^{13} size

- total: $8(2^{10} \cdot 2^{10} \cdot 1 \cdot 2^{13}) \neq$ 8256 KB ← all around the most convenient

- 3 level

L1	L2	L3	offset
7	8	8	13

- 1 page = 2^{13} bytes
- 1 L1 page = 2^8 pages
- 1 L2 page = 2^8 L3 pages

8 GB = 2^{33} B so 2^{20} pages, 2^{12} L3 pages, 2^9 L2 pages and 1 L1 page

$8(2^{12} \cdot 2^8 + 2^9 \cdot 2^8 + 2^7) =$ 8225 KB

3) Count the number of TLB misses

suppose the following code fits in one page and the stack also fits in one page. When the function is called the stack and code page are already in the TLB cache. Suppose an integer is 4Bytes long, a page contains 4096 B (2^{12})

```
int a[1024][1024], b... c...
multiply {
    unsigned i,j,k;
    for (i=0; i<1024; i++){
        for(j=0; j<1024; j++){
            for(k=0; k<1024; k++){
                c[i][j] += a[i][k] * b[k][j]
}}}
```

Given that the TLB cache contains only 8 entries and that it uses an LRU replacement policy, what is the number of TLB misses happening

- stack and code are always in TLB cache, so we have 6 entries left.
- each array consists of 1024 pages (each row)
  a page contains 4096 B, 1 int -> 4B $1024 \cdot 4 = 4096$

- determine i

  - requests: a[i][0], b[0][0], c[i][0], a[i][1], b[1][0], c[i][0]
  - all in a single page of a and c
  - 1 miss for a and 1 for c
  - b changes page every time 1024^2 misses
  - total: $2 + 2^{20}$ misses
  - ↳ for all i $1024(2 + 1024^2) = 2048 + 1024^3 = 2^{11} + 2^{30}$ misses

6) Inverted vs normal page tables

virt addresses 64 bits long, page size 1K Byte, physical memory size 64M Bytes
PTE takes up two bytes.
What is the total size of the inverted and standard page table? (Byte addressed)
tip: first estimate the number of Bytes an inverted page table needs, by checking which fields should at least be included and how many bits these should each contain

- standard
  - 64 bit : 2^{64} Byte addressable(?)
  - $\frac{2^{64}}{1K} = 2^{54}$ entries
  - $2^{54} \cdot 2B = 32768$ TBytes

- inverted
  - $\frac{64M}{1K} = \frac{2^{26}}{2^{10}} = 2^{16}$ inverted page table entries

how many Bytes for an entry?
- virtual page: 2^{54} entries: -> 54 bits
- pointer to other entry : 2^{16} entries -> 16 bits
- process id and flags : 10 bits (arbitrary choice)

} 80 bits ⇓ 10 Bytes

$2^{16} \cdot 10 = 640$ KByte

7)

Consider a system with 16KB of memory, 1KB pages, an inverted page table and a full-associative TLB with 2 entries and an LRU replacement policy.
The hash function used is f (PID, V P N) = (10 * PID + V P N) mod 24. The system needs 0.5ns to read the TLB, 100ns to read the main memory and 10ms to read the hard drive advice.
Complete the table below with page requests using the hash anchor table and the inverted page table (IPT).
In the IPT, "T" is used as a symbol to indicate that there is no next page table entry (PTE).
In the TLB column you display the contents of the TLB after the translation has taken place.
To do this, use the notation "(VPN, Frame number)" for each line in the cache, use "-" if the line in question is empty.
In the columns called "Frame number", "TLB hits" and "Page fault" you indicate respectively the physical frame number, whether there was a TLB hit during the translation and whether it led to a page fault or not.
Finally, enter the time it took to complete the translation in the last column.
We assume that the buffer can be updated in parallel after translating and executing the next instruction, so you don't have to take any time for this. In the event of any page faults, you do not need to adjust the data structures, but you do need to allow the time it takes to discover that there is a page fault.

erachter te komen dat er een page fault is.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Value	7	3	12	15	13	6	2	4	0	14	8	5	15	11	10	8	5	7	9	10	12	6	13	1

Tabel 1: Hash anchor table

IPT

≠0 =0

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PID	0	3	2	3	0	8	0	2	2	0	8	6	0	2	2	4
VPN	8	17	8	4	1	3	5	4	2	3	13	1	2	32	0	4
Next PTE	2	T	T	4	T	T	10	12	3	T	0	9	14	8	T	11

Tabel 2: Inverted Page table

No next PTE

f

	PID	VPN	TLB	Frame number	TLB hits	Page fault	Latency (ns)
1	0	1	(1,4) - /	4	~~Hit~~/Miss	~~Yes~~/No	0.5 + 100.3 = 300.5
2	0	2	(1,4) / (2,12)	12	~~Hit~~/Miss	~~Yes~~/No	200.5
1	0	1	(1,4) / (2,12)	4	Hit/~~Miss~~	~~Yes~~/No	0.5
3	0	3	(1,4) / (3,9)	9	~~Hit~~/Miss	~~Yes~~/No	400.5
11	8	3	(3,5) - /	5	~~Hit~~/Miss	~~Yes~~/No	200.5
3	0	3	(1,4) - /	9	~~Hit~~/Miss	~~Yes~~/No	400.5
22	2	2	(2,8) - /	8	~~Hit~~/Miss	~~Yes~~/No	300.5
21	2	1	(2,8) - /	/	~~Hit~~/Miss	Yes/~~No~~	500.5
21	8	13	(8,10) - /	10	~~Hit~~/Miss	~~Yes~~/No	300.5
14	8	3	(8,10) - (3,5)	5	~~Hit~~/Miss	~~Yes~~/No	200.5

flush cache

flush

flush 22 mod 24

flush

1 for T1, 2 for T2

8) count number of page faults

consider the following sequence of pages: 20 numbers

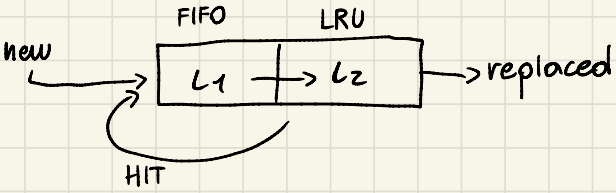
1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6

How many page faults occur if this process over 1,2,... 6 or 7 frames can be available
check with LRU, FIFO and MIN replacement algorithms.
Initially no page in memory

frames	LRU	FIFO	MIN
1	20	20	
2	18	18	
3	15	16	
4	10	14	
5	8	12	
6	7	10	
7	7	7	

1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6

9) Segmented FIFO



1) All pages in L_1 have been requested more recently than those in L_2

False:

L_1	A, B
L_2	C, D

request B, X =>

L_1	X, A
L_2	B, C

B has been requested more recently than A

2) Pages in L_1 are in LRU order
False, they are FIFO

3) Pages in L_2 are FIFO order
False, they are LRU order

4) $|L_2|-1$ most recently used pages are safe (cannot be replaced)
True

5) $|L_2|$ pages are safe
False, the first one in L_2 can be replaced

6) a seg. FIFO system with $|L_1|=|L_2|=2$ is equivalent to tree based pseudo LRU system with 4 Pages
False: with tree-based 1 page can be protected by a neighbor. Not possible here