© 2016 - 2017 — Rev: 10 – 13-Oct-2017. Silakan mengubah, memperbanyak, serta mendistribusikan dokumen ini selama tidak menghapus ketentuan ini. URL: http://rms46.vlsm.org/2/200.pdf

1. **2016-1 (McGill Fall 1998)**

   Asumsikan:

   i. Arsitektur komputer dengan ukuran halaman (page size) 1024 bytes.
   ii. Setiap karakter (char) menempati 1 alamat memori @ 1 byte.
   iii. Struktur data "matrix (baris,kolom)" untuk selanjutnya disebut "matrix".
   iv. Setiap 4 baris (berurutan) "matrix" berada dalam satu halaman (page).
   v. Setiap saat, maksimum ada 1 halaman (page) "matrix" dalam memori.
   vi. Saat awal eksekusi fungsi, tidak ada halaman (page) "matrix" dalam memori.

   Lingkari atau beri silang huruf "**B**" jika betul, dan "**S**" jika salah.

   B / S Fragmentasi eksternal (external fragmentation) akan terjadi pada sistem berbasis halaman (paging systems).

   B / S Bingkai (frame) pada memori virtual (VM) dipetakan ke halaman (page) pada memori fisik.

   B / S Pengeksekusian program berbasis demand paging selalu menghasilkan page fault.

   B / S Sebuah fungsi, mungkin saja menempati lebih dari satu halaman (page).

```
011 void isiMatrix1 (){
012    char matrix[256][256];
013    int ii, jj;
014    for (ii=0; ii<8; ii++) {
015       for (jj=0; jj<256; jj++) {
016          matrix[ii, jj] = 'x';
017       }
018    }
019 }
```

   B / S Setiap eksekusi baris 016, selalu akan terjadi "page fault" pada matrix.

   B / S Pada seluruh iterasi loop luar baris 014-018, akan terjadi 8 kali "page fault" pada matrix.

   B / S Pada saat mengeksekusi fungsi isiMatrix1(), terdapat kemungkinan terjadi TOTAL[1] lebih dari 3 kali "page fault".

```
021 void isiMatrix2 (){
022    char matrix[256][256];
023    int ii, jj;
024    for (jj=0; jj<256; jj++) {
025       for (ii=0; ii<8; ii++) {
026          matrix[ii, jj] = 'x';
027       }
028    }
029 }
```

   B / S Terdapat kemungkinan terjadi TOTAL 2 kali "page fault" saat mengeksekusi baris 026.

   B / S Pada setiap iterasi loop dalam baris 025-027, terjadi 2 kali "page fault" pada matrix.

   B / S Pada seluruh iterasi loop luar baris 024-028, terjadi 512 kali "page fault" pada matrix.

2. **2016-2 (Waterloo 2012)**

   Page Table.

   Consider this following "structure addrspace" of a 32-bit processor.

```
struct addrspace {
   vaddr_t as_vbase1     = 0x00100000; /* text  segment: virtual  base addr */
   paddr_t as_pbase1     = 0x10000000; /* text  segment: physical base addr */
   size_t  as_npages1    = 0x20;       /* text  segment: number   of  pages */
   vaddr_t as_vbase2     = 0x00200000; /* data  segment: virtual  base addr */
   paddr_t as_pbase2     = 0x20000000; /* data  segment: physical base addr */
   size_t  as_npages2    = 0x20;       /* data  segment: number   of  pages */
   vaddr_t as_vbase3     = 0x80000000; /* stack segment: virtual  base addr */
   paddr_t as_pbase3     = 0x80000000; /* stack segment: physical base addr */
   size_t  as_npages3    = 0x10;       /* stack segment: number   of  pages */
   int     page_size     = 0x1000;     /* virtual page size is 0x1000 bytes */
};
```

   When possible, translate the provided address.

Possible	Virtual Address	Physical Address	Segment
YES	**0x0010 0000**	**0x1000 0000**	**text**
NO	**0x0030 0000**	—	—
	0x0010 FEDC		
	0x0011 0000		
	0x7FFF FFFF		
		0x2000 1234	
		0x8000 FFFF	

3. **2017-1**

   (a) Please write down your student ID (NPM):

   |_____|_____ |_____|_____ |_____|_____ |_____|_____ |_____|_____ |

   (b) Please write down the last 2 digits of your student ID (NPM):

   |_____|_____ |

   (c) Please convert the 2 decimal digits above into an unsigned 32-bit hexdecimal number.

   Let's call that number **INTEGER32**: (**HEX**) |_____|_____ |_____|_____ |_____|_____ |_____|_____ |

   (d) Please add **INTEGER32** to 0080 0000 (HEX).

   Let's call it Virtual Address **ADDRESS32**: (**HEX**) |_____|_____ |_____|_____ |_____|_____ |_____|_____ |

   (e) **ADDRESS32** with a 4 kbyte page size will have page offset space of _____ **bits**,

   (f) And the page number space of **ADDRESS32** is _____ **bits**.

(g) Therefore, the page number of **ADDRESS32** is (**HEX**) _____,

(h) And, the page offset of **ADDRESS32** is (**HEX**) _____.

(i) The Page Table Entry (PTE) starts at Physical Address (PA) 001 000 (HEX). Each PTE consists of 4 hexadecimal digits (16 bits or 2 bytes) which is stored in BIG-ENDIAN form.

PA (HEX)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
001 000	71	00	71	03	71	05	71	07	71	09	71	0B	71	0D	71	0F
001 010	71	10	71	13	71	15	71	17	71	19	71	1B	71	1D	71	1F
...																
002 000	71	20	71	23	71	25	71	27	71	29	71	2B	71	2D	71	2F
002 010	71	30	71	33	71	35	71	37	71	39	71	3B	71	3D	71	3F
...																
003 000	71	40	71	43	71	45	71	47	71	49	71	4B	71	4D	71	4F
003 010	71	41	71	53	71	55	71	57	71	59	71	5B	71	5D	71	5D
...																

The PTE of page number ADDRESS32 is: (**HEX**) _____.

(j) The first digit are the flags. The PTE is valid when the flags digit is not zero (0). The flags of the PTE above is (**HEX**) _____ which means the PTE is (VALID / NOT VALID).

(k) If the PTE is VALID, the next three digits are the Physical Frame Number: (**HEX**) _____.

(l) Thus, the physical address of ADDRESS32 is: (**HEX**) _____.

(m) Please put INTEGER32 into the physical address of ADDRESS32 (BIG-ENDIAN form).

-PA- (HEX)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F