第 20 章

1.对于线性页表，你需要一个寄存器来定位页表，假设硬件在 TLB 未命中时进行查找。你需要多少个寄存器才能找到两级页表？三级页表呢？

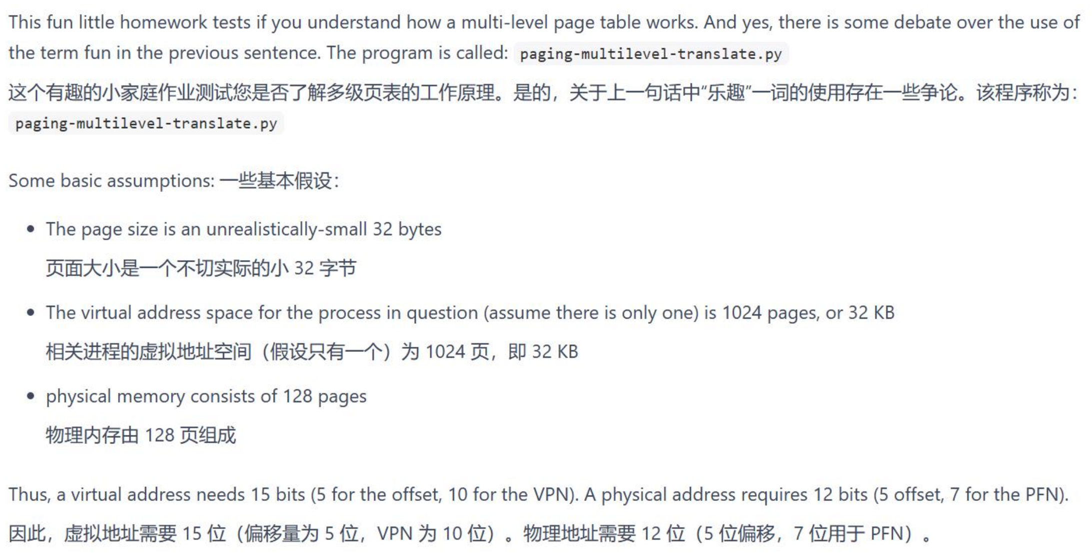
对于二级页表，需要通过寄存器找到页目录中的存放页表的位置，然后从页表中找到对应页表项的位置。
对于三级页表，需要通过寄存器找到一级页目录中存放二级页目录的位置，然后从二级页目录中寻找存放页表的位置，最后从页表中找到对应页表项的位置。

这里我觉得需要考虑页目录以及页表的页表项规模，来判断需要用多少寄存器，比如 32 位系统下，
二级页表中每一级的页表项分别为 2 的 L1 次方个，2 的 L2 次方个，总共需要(log2(L1) + log2(L2) + OFFSET 的位数)/32 （向上取整）个寄存器来定位页表

三级页表中每一级的页表项分别为 2 的 L1 次方个，2 的 L2 次方个，2 的 L3 次方个，总共需要(log2(L1) + log2(L2) + log2(L3) + OFFSET 的位数)/32 （向上取整）个寄存器来定位页表

2. 使用模拟器对随机种子 0、1 和 2 执行翻译，并使用-c 标志检查你的答案。需要多少内存引用来执行每次查找？

首先阅读相关文档



This fun little homework tests if you understand how a multi-level page table works. And yes, there is some debate over the use of the term fun in the previous sentence. The program is called: `paging-multilevel-translate.py`

这个有趣的小家庭作业测试您是否了解多级页表的工作原理。是的，关于上一句话中“乐趣”一词的使用存在一些争论。该程序称为：`paging-multilevel-translate.py`

Some basic assumptions: 一些基本假设：

- The page size is an unrealistically-small 32 bytes
  页面大小是一个不切实际的小 32 字节
- The virtual address space for the process in question (assume there is only one) is 1024 pages, or 32 KB
  相关进程的虚拟地址空间（假设只有一个）为 1024 页，即 32 KB
- physical memory consists of 128 pages
  物理内存由 128 页组成

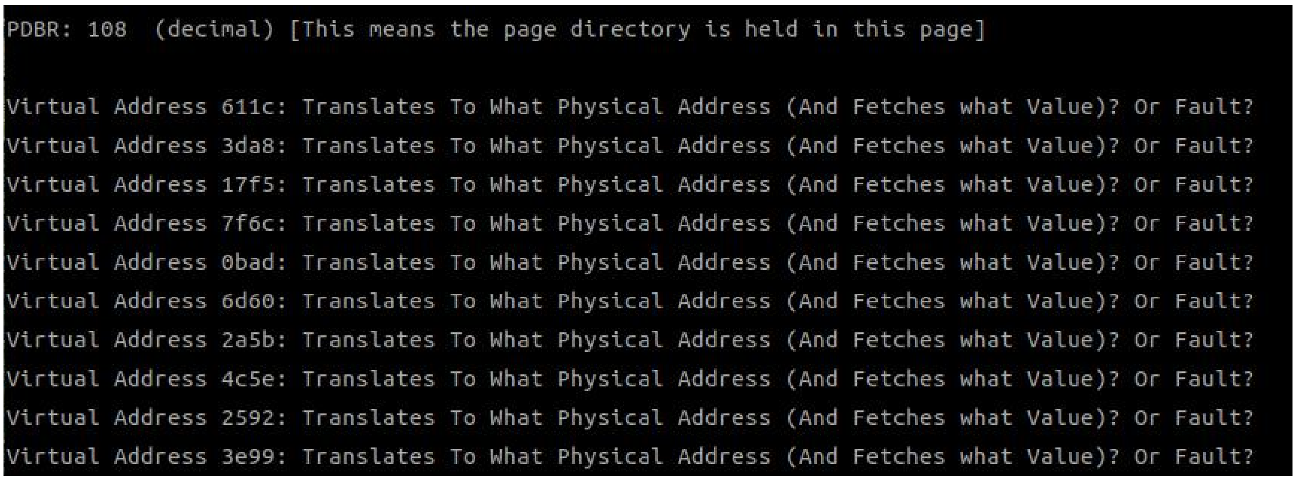
Thus, a virtual address needs 15 bits (5 for the offset, 10 for the VPN). A physical address requires 12 bits (5 offset, 7 for the PFN).

因此，虚拟地址需要 15 位（偏移量为 5 位，VPN 为 10 位）。物理地址需要 12 位（5 位偏移，7 位用于 PFN）。

使用以下命令：

```
python3 paging-multilevel-translate.py -s 0
python3 paging-multilevel-translate.py -s 1
python3 paging-multilevel-translate.py -s 2
```

种子 0：



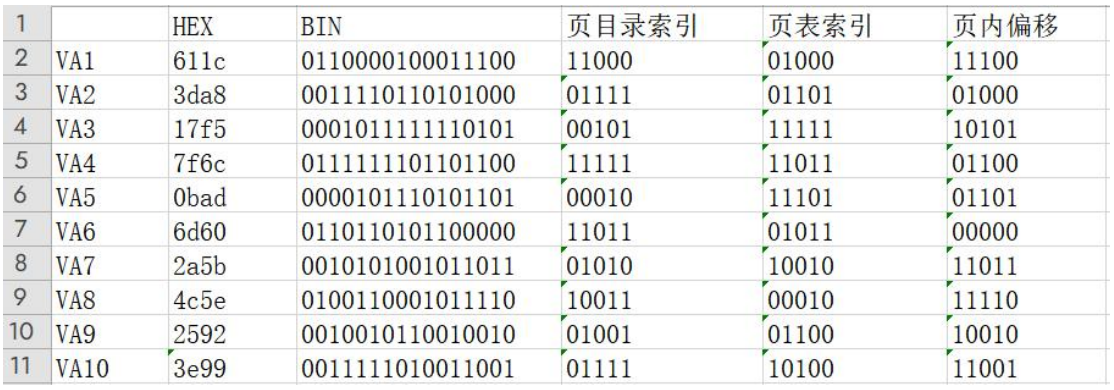
```
PDBR: 108  (decimal) [This means the page directory is held in this page]

Virtual Address 611c: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 3da8: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 17f5: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 7f6c: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 0bad: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 6d60: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 2a5b: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 4c5e: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 2592: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 3e99: Translates To What Physical Address (And Fetches what Value)? Or Fault?
```

VA 0：

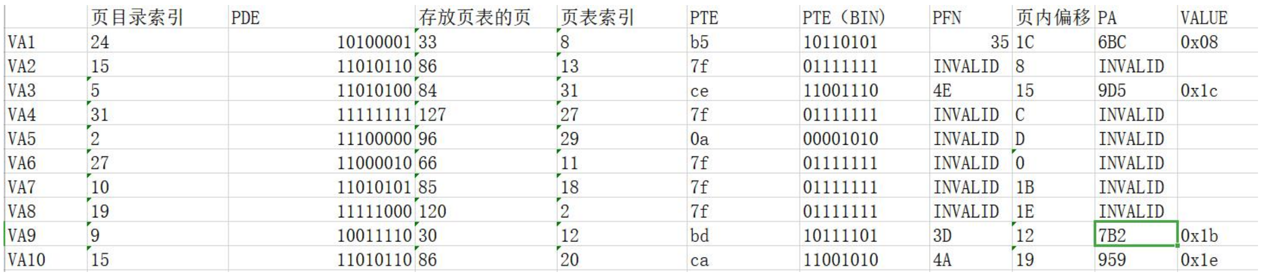
611C　110000100011100，页目录索引：11000（24），页表索引：01000（8），页内偏移：11100（28），页目录基址：108。

首先找到页目录项，在 108 页的 25 字节，为 0xa1（10100001），最高位表明有效，后七位为页表所在的页：33。

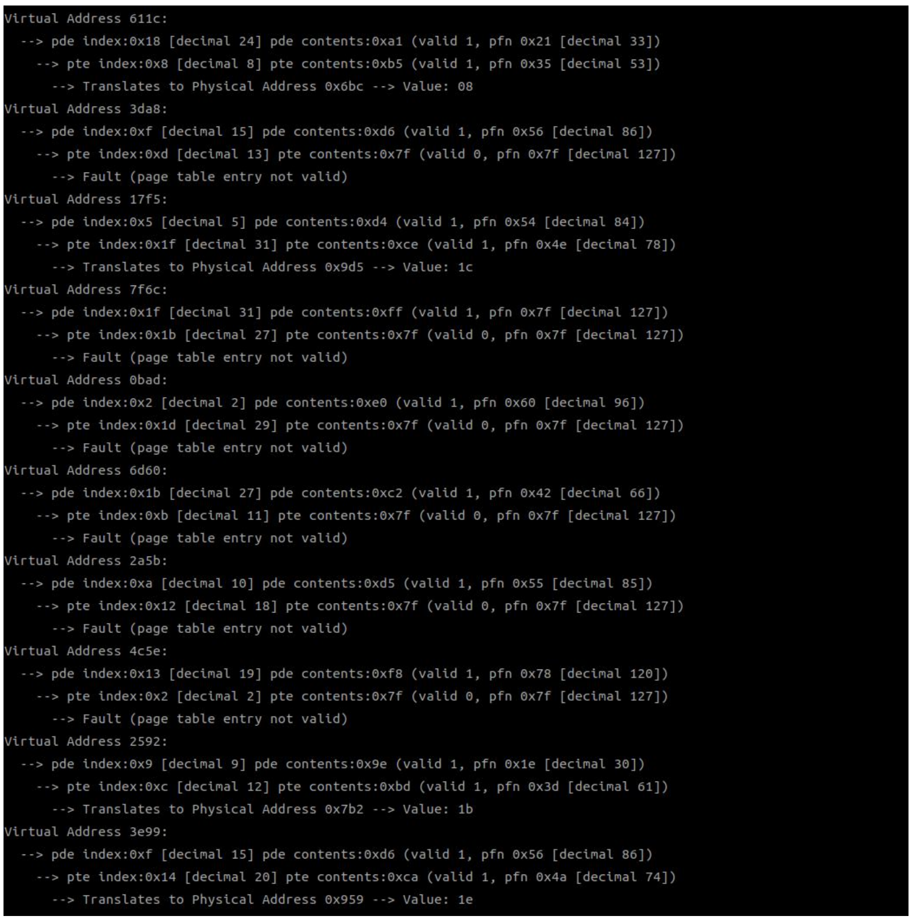
页表索引为 8，即 33 页的 9 字节，为 0xb5（10110101），最高位表明有效，后七位为物理页帧 0x35（53），则物理地址为 PA=53*32+28=1724（0x6bc），找到 35 页的第 29 个字节即为最终找到的值：0x08



1		HEX	BIN	页目录索引	页表索引	页内偏移
2	VA1	611c	0110000100011100	11000	01000	11100
3	VA2	3da8	0011110110101000	01111	01101	01000
4	VA3	17f5	0001011111110101	00101	11111	10101
5	VA4	7f6c	0111111101101100	11111	11011	01100
6	VA5	0bad	0000101110101101	00010	11101	01101
7	VA6	6d60	0110110101100000	11011	01011	00000
8	VA7	2a5b	0010101001011011	01010	10010	11011
9	VA8	4c5e	0100110001011110	10011	00010	11110
10	VA9	2592	0010010110010010	01001	01100	10010
11	VA10	3e99	0011111010011001	01111	10100	11001



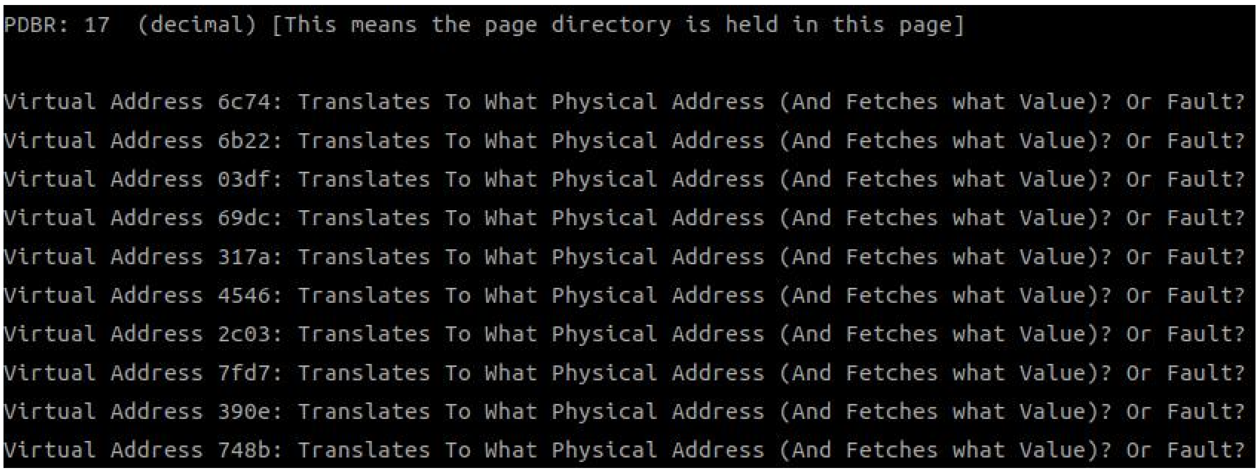
	页目录索引	PDE	存放页表的页	页表索引	PTE	PTE（BIN）	PFN	页内偏移	PA	VALUE
VA1	24	10100001	33	8	b5	10110101	35	1C	6BC	0x08
VA2	15	11010110	86	13	7f	01111111	INVALID	8	INVALID	
VA3	5	11010100	84	31	ce	11001110	4E	15	9D5	0x1c
VA4	31	11111111	127	27	7f	01111111	INVALID	C	INVALID	
VA5	2	11100000	96	29	0a	00001010	INVALID	D	INVALID	
VA6	27	11000010	66	11	7f	01111111	INVALID	0	INVALID	
VA7	10	11010101	85	18	7f	01111111	INVALID	1B	INVALID	
VA8	19	11111000	120	2	7f	01111111	INVALID	1E	INVALID	
VA9	9	10011110	30	12	bd	10111101	3D	12	7B2	0x1b
VA10	15	11010110	86	20	ca	11001010	4A	19	959	0x1e



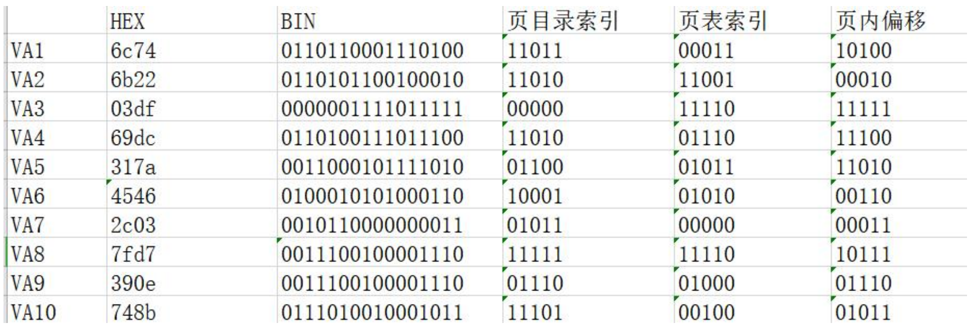
```
Virtual Address 611c:
  --> pde index:0x18 [decimal 24] pde contents:0xa1 (valid 1, pfn 0x21 [decimal 33])
    --> pte index:0x8 [decimal 8] pte contents:0xb5 (valid 1, pfn 0x35 [decimal 53])
      --> Translates to Physical Address 0x6bc --> Value: 08
Virtual Address 3da8:
  --> pde index:0xf [decimal 15] pde contents:0xd6 (valid 1, pfn 0x56 [decimal 86])
    --> pte index:0xd [decimal 13] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 17f5:
  --> pde index:0x5 [decimal 5] pde contents:0xd4 (valid 1, pfn 0x54 [decimal 84])
    --> pte index:0x1f [decimal 31] pte contents:0xce (valid 1, pfn 0x4e [decimal 78])
      --> Translates to Physical Address 0x9d5 --> Value: 1c
Virtual Address 7f6c:
  --> pde index:0x1f [decimal 31] pde contents:0xff (valid 1, pfn 0x7f [decimal 127])
    --> pte index:0x1b [decimal 27] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 0bad:
  --> pde index:0x2 [decimal 2] pde contents:0xe0 (valid 1, pfn 0x60 [decimal 96])
    --> pte index:0x1d [decimal 29] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 6d60:
  --> pde index:0x1b [decimal 27] pde contents:0xc2 (valid 1, pfn 0x42 [decimal 66])
    --> pte index:0xb [decimal 11] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 2a5b:
  --> pde index:0xa [decimal 10] pde contents:0xd5 (valid 1, pfn 0x55 [decimal 85])
    --> pte index:0x12 [decimal 18] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 4c5e:
  --> pde index:0x13 [decimal 19] pde contents:0xf8 (valid 1, pfn 0x78 [decimal 120])
    --> pte index:0x2 [decimal 2] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 2592:
  --> pde index:0x9 [decimal 9] pde contents:0x9e (valid 1, pfn 0x1e [decimal 30])
    --> pte index:0xc [decimal 12] pte contents:0xbd (valid 1, pfn 0x3d [decimal 61])
      --> Translates to Physical Address 0x7b2 --> Value: 1b
Virtual Address 3e99:
  --> pde index:0xf [decimal 15] pde contents:0xd6 (valid 1, pfn 0x56 [decimal 86])
    --> pte index:0x14 [decimal 20] pte contents:0xca (valid 1, pfn 0x4a [decimal 74])
      --> Translates to Physical Address 0x959 --> Value: 1e
```

可以看到结果和我们预测的一致

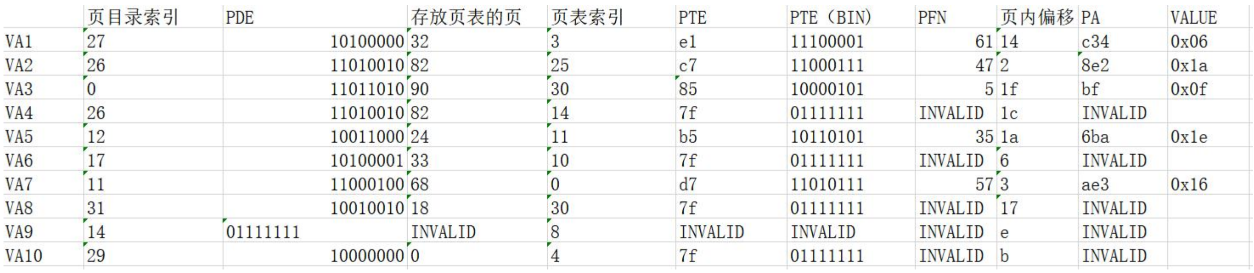
种子 1：



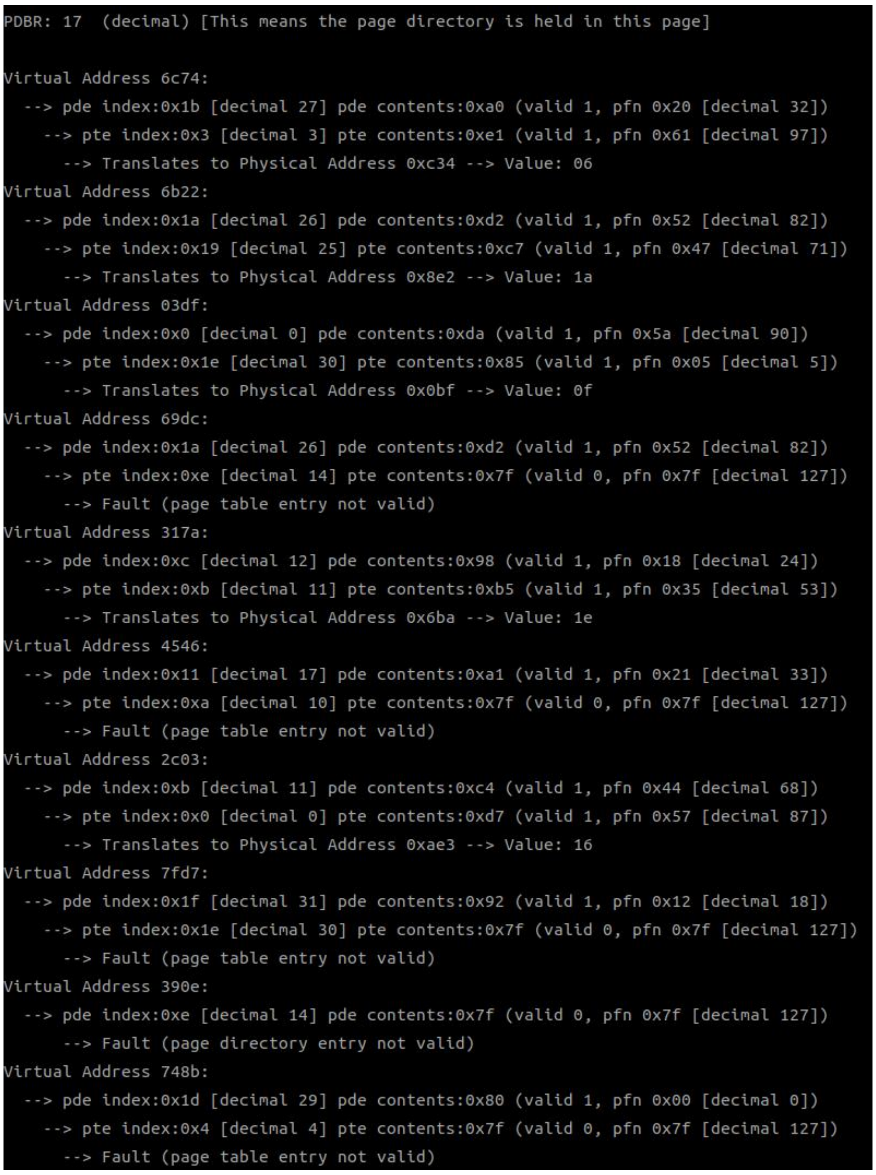
```
PDBR: 17  (decimal) [This means the page directory is held in this page]

Virtual Address 6c74: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 6b22: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 03df: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 69dc: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 317a: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 4546: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 2c03: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 7fd7: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 390e: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 748b: Translates To What Physical Address (And Fetches what Value)? Or Fault?
```



	HEX	BIN	页目录索引	页表索引	页内偏移
VA1	6c74	0110110001110100	11011	00011	10100
VA2	6b22	0110101100100010	11010	11001	00010
VA3	03df	0000001111011111	00000	11110	11111
VA4	69dc	0110100111011100	11010	01110	11100
VA5	317a	0011000101111010	01100	01011	11010
VA6	4546	0100010101000110	10001	01010	00110
VA7	2c03	0010110000000011	01011	00000	00011
VA8	7fd7	0011100100001110	11111	11110	10111
VA9	390e	0011100100001110	01110	01000	01110
VA10	748b	0111010010001011	11101	00100	01011



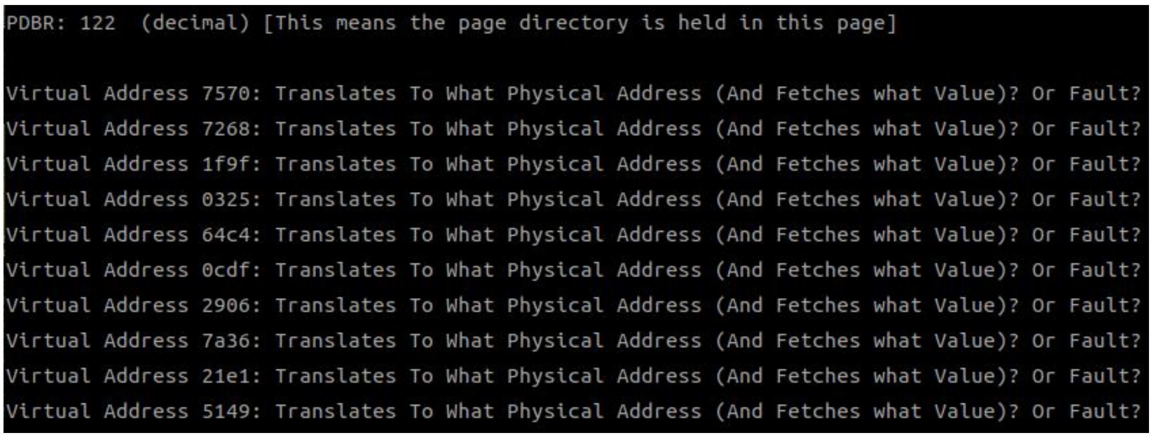
	页目录索引	PDE	存放页表的页	页表索引	PTE	PTE（BIN）	PFN	页内偏移	PA	VALUE
VA1	27	10100000	32	3	e1	11100001	61	14	c34	0x06
VA2	26	11010010	82	25	c7	11000111	47	2	8e2	0x1a
VA3	0	11011010	90	30	85	10000101	5	1f	bf	0x0f
VA4	26	11010010	82	14	7f	01111111	INVALID	1c	INVALID	
VA5	12	10011000	24	11	b5	10110101	35	1a	6ba	0x1e
VA6	17	10100001	33	10	7f	01111111	INVALID	6	INVALID	
VA7	11	11000100	68	0	d7	11010111	57	3	ae3	0x16
VA8	31	10010010	18	30	7f	01111111	INVALID	17	INVALID	
VA9	14	01111111	INVALID	8	INVALID	INVALID	INVALID	e	INVALID	
VA10	29	10000000	0	4	7f	01111111	INVALID	b	INVALID	



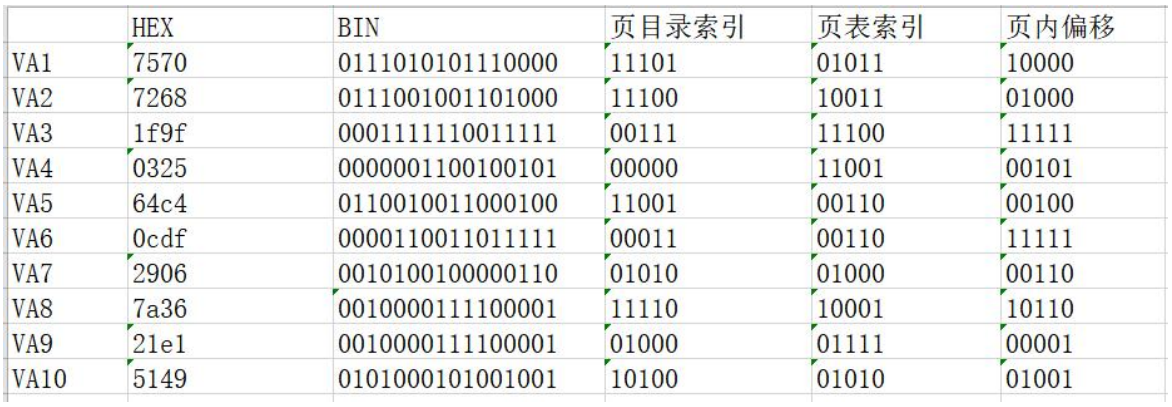
```
PDBR: 17  (decimal) [This means the page directory is held in this page]

Virtual Address 6c74:
  --> pde index:0x1b [decimal 27] pde contents:0xa0 (valid 1, pfn 0x20 [decimal 32])
    --> pte index:0x3 [decimal 3] pte contents:0xe1 (valid 1, pfn 0x61 [decimal 97])
      --> Translates to Physical Address 0xc34 --> Value: 06
Virtual Address 6b22:
  --> pde index:0x1a [decimal 26] pde contents:0xd2 (valid 1, pfn 0x52 [decimal 82])
    --> pte index:0x19 [decimal 25] pte contents:0xc7 (valid 1, pfn 0x47 [decimal 71])
      --> Translates to Physical Address 0x8e2 --> Value: 1a
Virtual Address 03df:
  --> pde index:0x0 [decimal 0] pde contents:0xda (valid 1, pfn 0x5a [decimal 90])
    --> pte index:0x1e [decimal 30] pte contents:0x85 (valid 1, pfn 0x05 [decimal 5])
      --> Translates to Physical Address 0x0bf --> Value: 0f
Virtual Address 69dc:
  --> pde index:0x1a [decimal 26] pde contents:0xd2 (valid 1, pfn 0x52 [decimal 82])
    --> pte index:0xe [decimal 14] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 317a:
  --> pde index:0xc [decimal 12] pde contents:0x98 (valid 1, pfn 0x18 [decimal 24])
    --> pte index:0xb [decimal 11] pte contents:0xb5 (valid 1, pfn 0x35 [decimal 53])
      --> Translates to Physical Address 0x6ba --> Value: 1e
Virtual Address 4546:
  --> pde index:0x11 [decimal 17] pde contents:0xa1 (valid 1, pfn 0x21 [decimal 33])
    --> pte index:0xa [decimal 10] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 2c03:
  --> pde index:0xb [decimal 11] pde contents:0xc4 (valid 1, pfn 0x44 [decimal 68])
    --> pte index:0x0 [decimal 0] pte contents:0xd7 (valid 1, pfn 0x57 [decimal 87])
      --> Translates to Physical Address 0xae3 --> Value: 16
Virtual Address 7fd7:
  --> pde index:0x1f [decimal 31] pde contents:0x92 (valid 1, pfn 0x12 [decimal 18])
    --> pte index:0x1e [decimal 30] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 390e:
  --> pde index:0xe [decimal 14] pde contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page directory entry not valid)
Virtual Address 748b:
  --> pde index:0x1d [decimal 29] pde contents:0x80 (valid 1, pfn 0x00 [decimal 0])
    --> pte index:0x4 [decimal 4] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
```

结果和我们计算的一致

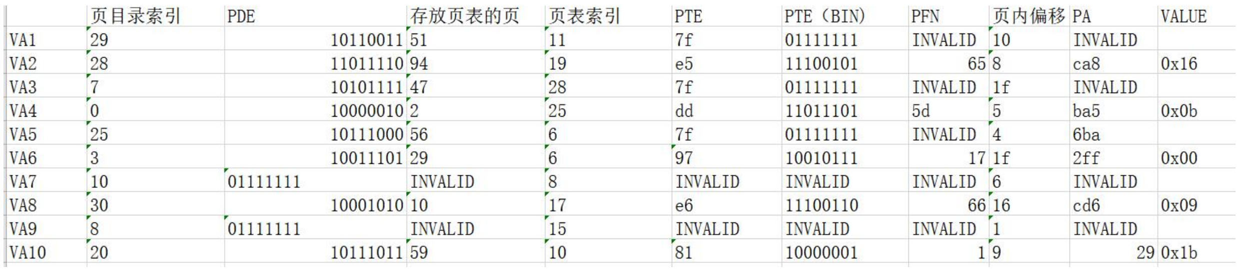
种子 2：



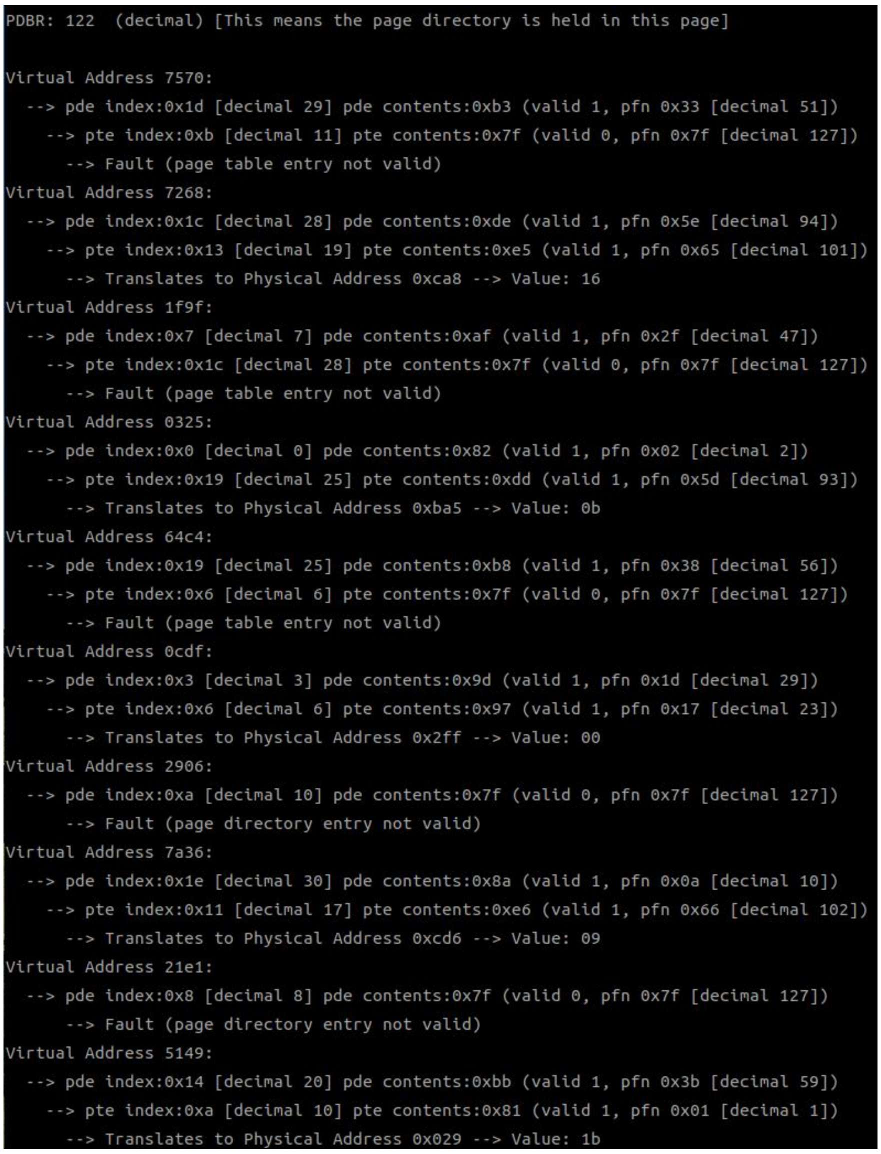
```
PDBR: 122  (decimal) [This means the page directory is held in this page]

Virtual Address 7570: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 7268: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 1f9f: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 0325: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 64c4: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 0cdf: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 2906: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 7a36: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 21e1: Translates To What Physical Address (And Fetches what Value)? Or Fault?
Virtual Address 5149: Translates To What Physical Address (And Fetches what Value)? Or Fault?
```



	HEX	BIN	页目录索引	页表索引	页内偏移
VA1	7570	0111010101110000	11101	01011	10000
VA2	7268	0111001001101000	11100	10011	01000
VA3	1f9f	0001111110011111	00111	11100	11111
VA4	0325	0000001100100101	00000	11001	00101
VA5	64c4	0110010011000100	11001	00110	00100
VA6	0cdf	0000110011011111	00011	00110	11111
VA7	2906	0010100100000110	01010	01000	00110
VA8	7a36	0010000111100001	11110	10001	10110
VA9	21e1	0010000111100001	01000	01111	00001
VA10	5149	0101000101001001	10100	01010	01001



	页目录索引	PDE	存放页表的页	页表索引	PTE	PTE（BIN）	PFN	页内偏移	PA	VALUE
VA1	29	10110011	51	11	7f	01111111	INVALID	10	INVALID	
VA2	28	11011110	94	19	e5	11100101	65	8	ca8	0x16
VA3	7	10101111	47	28	7f	01111111	INVALID	1f	INVALID	
VA4	0	10000010	2	25	dd	11011101	5d	5	ba5	0x0b
VA5	25	10111000	56	6	7f	01111111	INVALID	4	6ba	
VA6	3	10011101	29	6	97	10010111	17	1f	2ff	0x00
VA7	10	01111111	INVALID	8	INVALID	INVALID	INVALID	6	INVALID	
VA8	30	10001010	10	17	e6	11100110	66	16	cd6	0x09
VA9	8	01111111	INVALID	15	INVALID	INVALID	INVALID	1	INVALID	
VA10	20	10111011	59	10	81	10000001	1	9	29	0x1b



```
PDBR: 122  (decimal) [This means the page directory is held in this page]

Virtual Address 7570:
  --> pde index:0x1d [decimal 29] pde contents:0xb3 (valid 1, pfn 0x33 [decimal 51])
    --> pte index:0xb [decimal 11] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 7268:
  --> pde index:0x1c [decimal 28] pde contents:0xde (valid 1, pfn 0x5e [decimal 94])
    --> pte index:0x13 [decimal 19] pte contents:0xe5 (valid 1, pfn 0x65 [decimal 101])
      --> Translates to Physical Address 0xca8 --> Value: 16
Virtual Address 1f9f:
  --> pde index:0x7 [decimal 7] pde contents:0xaf (valid 1, pfn 0x2f [decimal 47])
    --> pte index:0x1c [decimal 28] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 0325:
  --> pde index:0x0 [decimal 0] pde contents:0x82 (valid 1, pfn 0x02 [decimal 2])
    --> pte index:0x19 [decimal 25] pte contents:0xdd (valid 1, pfn 0x5d [decimal 93])
      --> Translates to Physical Address 0xba5 --> Value: 0b
Virtual Address 64c4:
  --> pde index:0x19 [decimal 25] pde contents:0xb8 (valid 1, pfn 0x38 [decimal 56])
    --> pte index:0x6 [decimal 6] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 0cdf:
  --> pde index:0x3 [decimal 3] pde contents:0x9d (valid 1, pfn 0x1d [decimal 29])
    --> pte index:0x6 [decimal 6] pte contents:0x97 (valid 1, pfn 0x17 [decimal 23])
      --> Translates to Physical Address 0x2ff --> Value: 00
Virtual Address 2906:
  --> pde index:0xa [decimal 10] pde contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page directory entry not valid)
Virtual Address 7a36:
  --> pde index:0x1e [decimal 30] pde contents:0x8a (valid 1, pfn 0x0a [decimal 10])
    --> pte index:0x11 [decimal 17] pte contents:0xe6 (valid 1, pfn 0x66 [decimal 102])
      --> Translates to Physical Address 0xcd6 --> Value: 09
Virtual Address 21e1:
  --> pde index:0x8 [decimal 8] pde contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page directory entry not valid)
Virtual Address 5149:
  --> pde index:0x14 [decimal 20] pde contents:0xbb (valid 1, pfn 0x3b [decimal 59])
    --> pte index:0xa [decimal 10] pte contents:0x81 (valid 1, pfn 0x01 [decimal 1])
      --> Translates to Physical Address 0x029 --> Value: 1b
```

结果和我们计算的一致

3. 根据你对缓存内存的工作原理的理解，你认为对页表的内存引用如何在缓存中工作？它们是否会导致大量的缓存命中（并导致快速访问）或者很多未命中（并导致访问缓慢）？

初次访问内存中的某个位置时，会产生不命中，这个不命中是必然发生的（强制不命中）。

系统将访问页表，找到虚拟页所对应的物理页，并将这个映射保存到缓存中。利用时间局部性与空间局部性，接下来的访问很有可能导致大量的缓存命中，从而导致快速访问。

在一些特定的情况下，程序短时间访问的页数大于缓存中的页数，可能会产生大量的缓存不命中。

除此之外，缓存的替换算法和访问的行为模式也会造成一定的影响。

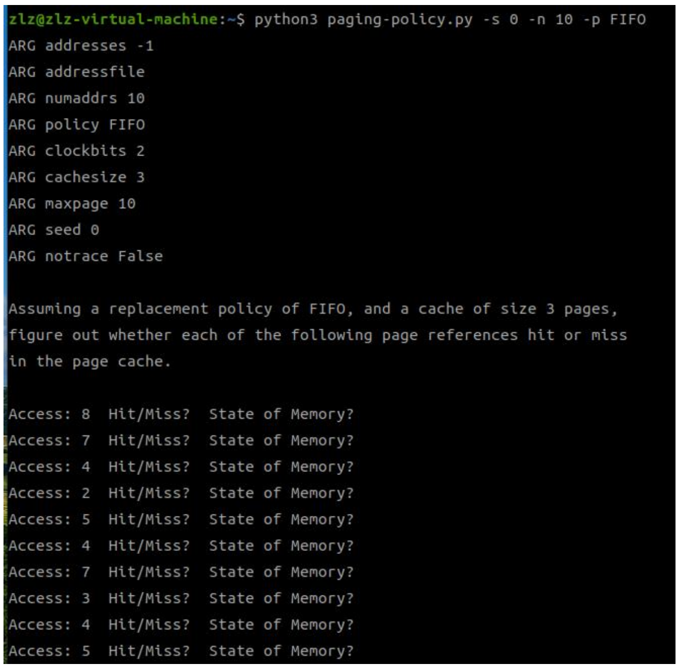
第 22 章

1.使用以下参数生成随机地址：-s 0 -n 10，-s 1 -n 10 和-s 2 -n 10。将策略从 FIFO 更改为 LRU，并将其更改为 OPT。计算所述地址追踪中的每个访问是否命中或未命中。

种子 0：

（1）FIFO

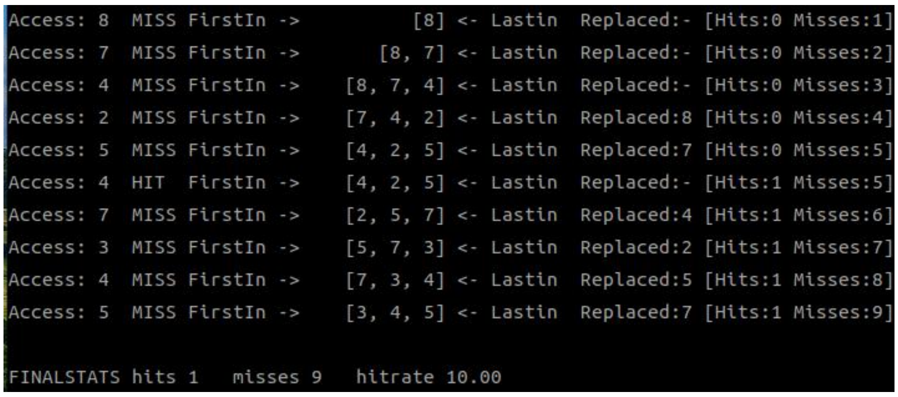
python3 paging-policy.py -s 0 -n 10 -p FIFO



```
zlz@zlz-virtual-machine:~$ python3 paging-policy.py -s 0 -n 10 -p FIFO
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Assuming a replacement policy of FIFO, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 2  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 3  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
```

```
Access 8:Miss  [8]     Lastin  Replaced:- [Hits:0 Misses:1]
Access 7:Miss  [8,7]   Lastin  Replaced:- [Hits:0 Misses:2]
Access 4:Miss  [8,7,4] Lastin  Replaced:- [Hits:0 Misses:3]
Access 2:Miss  [7,4,2] Lastin  Replaced:8 [Hits:0 Misses:4]
Access 5:Miss  [4,2,5] Lastin  Replaced:7 [Hits:0 Misses:5]
Access 4:Hit   [4,2,5] Lastin  Replaced:- [Hits:1 Misses:5]
Access 7:Miss  [2,5,7] Lastin  Replaced:4 [Hits:1 Misses:6]
Access 3:Miss  [5,7,3] Lastin  Replaced:2 [Hits:1 Misses:7]
Access 4:Miss  [7,3,4] Lastin  Replaced:5 [Hits:1 Misses:8]
Access 5:Miss  [3,4,5] Lastin  Replaced:7 [Hits:1 Misses:9]
```

命中率是 10%

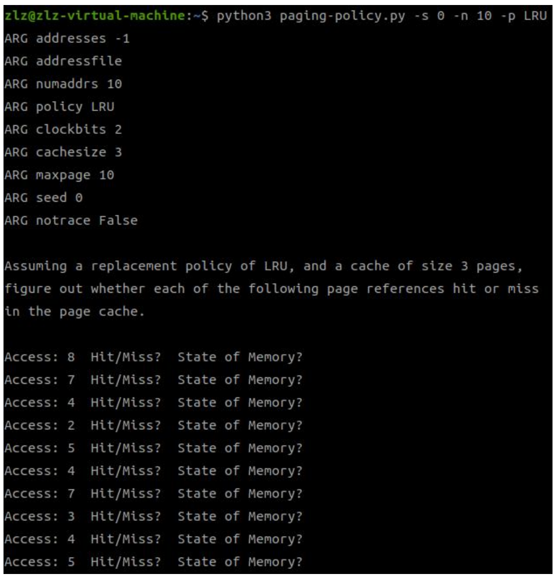


```
Access: 8  MISS FirstIn ->          [8] <- Lastin  Replaced:- [Hits:0 Misses:1]
Access: 7  MISS FirstIn ->       [8, 7] <- Lastin  Replaced:- [Hits:0 Misses:2]
Access: 4  MISS FirstIn ->    [8, 7, 4] <- Lastin  Replaced:- [Hits:0 Misses:3]
Access: 2  MISS FirstIn ->    [7, 4, 2] <- Lastin  Replaced:8 [Hits:0 Misses:4]
Access: 5  MISS FirstIn ->    [4, 2, 5] <- Lastin  Replaced:7 [Hits:0 Misses:5]
Access: 4  HIT  FirstIn ->    [4, 2, 5] <- Lastin  Replaced:- [Hits:1 Misses:5]
Access: 7  MISS FirstIn ->    [2, 5, 7] <- Lastin  Replaced:4 [Hits:1 Misses:6]
Access: 3  MISS FirstIn ->    [5, 7, 3] <- Lastin  Replaced:2 [Hits:1 Misses:7]
Access: 4  MISS FirstIn ->    [7, 3, 4] <- Lastin  Replaced:5 [Hits:1 Misses:8]
Access: 5  MISS FirstIn ->    [3, 4, 5] <- Lastin  Replaced:7 [Hits:1 Misses:9]

FINALSTATS hits 1   misses 9   hitrate 10.00
```

结果和我们预测的一致

（2）LRU

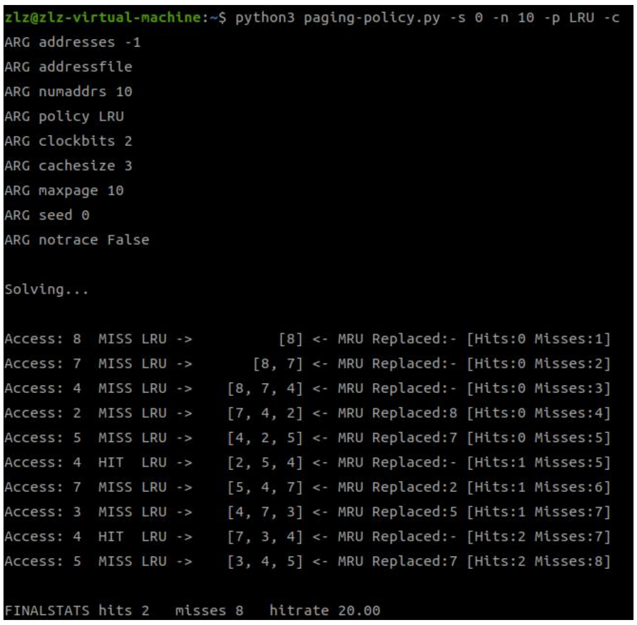
python3 paging-policy.py -s 0 -n 10 -p LRU



```
zlz@zlz-virtual-machine:~$ python3 paging-policy.py -s 0 -n 10 -p LRU
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Assuming a replacement policy of LRU, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 2  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 3  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
```

```
Access 8:Miss  [8]     Lastin  Replaced:- [Hits:0 Misses:1]
Access 7:Miss  [8,7]   Lastin  Replaced:- [Hits:0 Misses:2]
Access 4:Miss  [8,7,4] Lastin  Replaced:- [Hits:0 Misses:3]
Access 2:Miss  [7,4,2] Lastin  Replaced:8 [Hits:0 Misses:4]
Access 5:Miss  [4,2,5] Lastin  Replaced:7 [Hits:0 Misses:5]
Access 4:Hit   [4,2,5] Lastin  Replaced:- [Hits:1 Misses:5]
Access 7:Miss  [4,5,7] Lastin  Replaced:2 [Hits:1 Misses:6]
Access 3:Miss  [4,7,3] Lastin  Replaced:5 [Hits:1 Misses:7]
Access 4:Hit   [4,7,3] Lastin  Replaced:- [Hits:2 Misses:7]
Access 5:Miss  [4,3,5] Lastin  Replaced:7 [Hits:2 Misses:8]
```

命中率是 20%

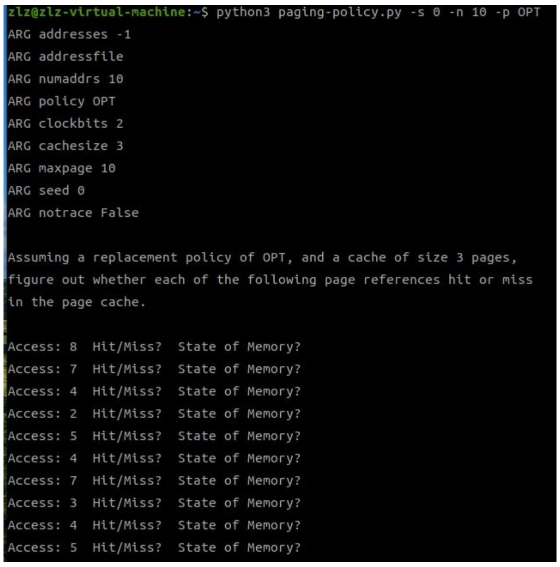


```
zlz@zlz-virtual-machine:~$ python3 paging-policy.py -s 0 -n 10 -p LRU -c
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 8  MISS LRU ->          [8] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 7  MISS LRU ->       [8, 7] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 4  MISS LRU ->    [8, 7, 4] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 2  MISS LRU ->    [7, 4, 2] <- MRU Replaced:8 [Hits:0 Misses:4]
Access: 5  MISS LRU ->    [4, 2, 5] <- MRU Replaced:7 [Hits:0 Misses:5]
Access: 4  HIT  LRU ->    [2, 5, 4] <- MRU Replaced:- [Hits:1 Misses:5]
Access: 7  MISS LRU ->    [5, 4, 7] <- MRU Replaced:2 [Hits:1 Misses:6]
Access: 3  MISS LRU ->    [4, 7, 3] <- MRU Replaced:5 [Hits:1 Misses:7]
Access: 4  HIT  LRU ->    [7, 3, 4] <- MRU Replaced:- [Hits:2 Misses:7]
Access: 5  MISS LRU ->    [3, 4, 5] <- MRU Replaced:7 [Hits:2 Misses:8]

FINALSTATS hits 2   misses 8   hitrate 20.00
```

结果和我们预测的一致

（3）OPT

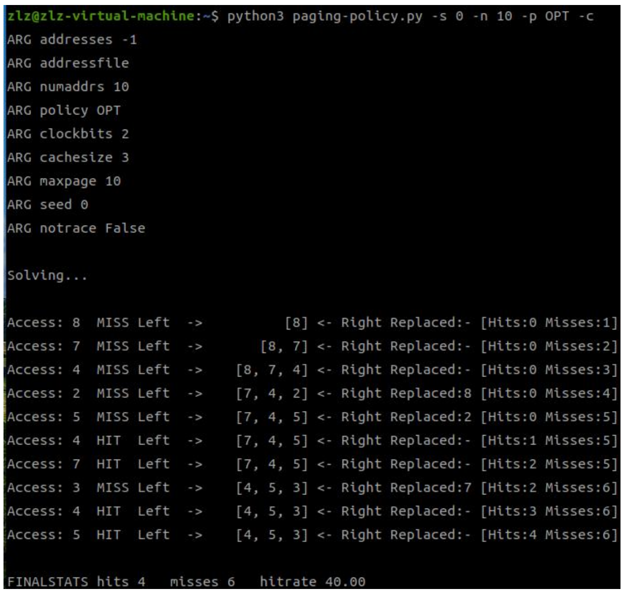
python3 paging-policy.py -s 0 -n 10 -p OPT



```
zlz@zlz-virtual-machine:~$ python3 paging-policy.py -s 0 -n 10 -p OPT
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy OPT
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Assuming a replacement policy of OPT, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 2  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 3  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
```

```
Access 8:Miss  [8]     Lastin  Replaced:- [Hits:0 Misses:1]
Access 7:Miss  [8,7]   Lastin  Replaced:- [Hits:0 Misses:2]
Access 4:Miss  [8,7,4] Lastin  Replaced:- [Hits:0 Misses:3]
Access 2:Miss  [7,4,2] Lastin  Replaced:8 [Hits:0 Misses:4]
Access 5:Miss  [7,4,5] Lastin  Replaced:2 [Hits:0 Misses:5]
Access 4:Hit   [7,4,5] Lastin  Replaced:- [Hits:1 Misses:5]
Access 7:Hit   [7,4,5] Lastin  Replaced:- [Hits:2 Misses:5]
Access 3:Miss  [4,5,3] Lastin  Replaced:7 [Hits:2 Misses:6]
Access 4:Hit   [4,5,3] Lastin  Replaced:- [Hits:3 Misses:6]
Access 5:Hit   [4,5,3] Lastin  Replaced:- [Hits:4 Misses:6]
```

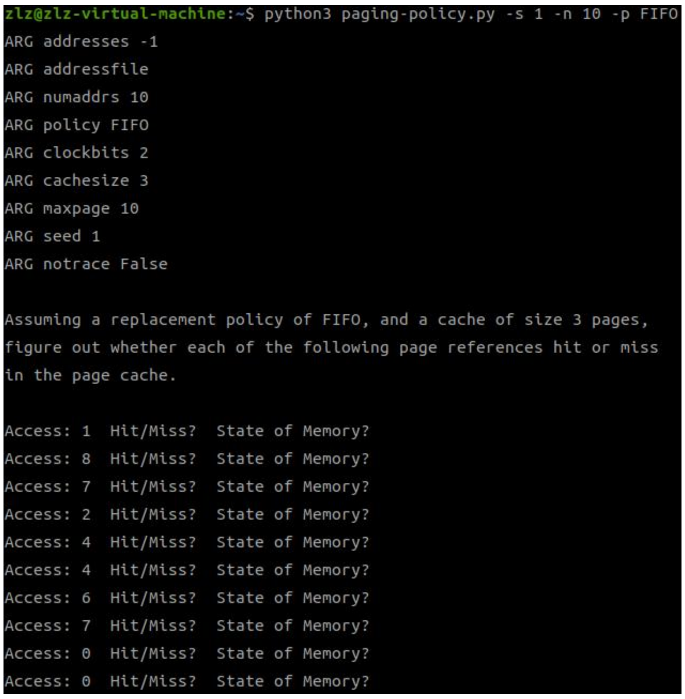
命中率是 40%



```
zlz@zlz-virtual-machine:~$ python3 paging-policy.py -s 0 -n 10 -p OPT -c
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy OPT
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 8  MISS Left  ->          [8] <- Right Replaced:- [Hits:0 Misses:1]
Access: 7  MISS Left  ->       [8, 7] <- Right Replaced:- [Hits:0 Misses:2]
Access: 4  MISS Left  ->    [8, 7, 4] <- Right Replaced:- [Hits:0 Misses:3]
Access: 2  MISS Left  ->    [7, 4, 2] <- Right Replaced:8 [Hits:0 Misses:4]
Access: 5  MISS Left  ->    [7, 4, 5] <- Right Replaced:2 [Hits:0 Misses:5]
Access: 4  HIT  Left  ->    [7, 4, 5] <- Right Replaced:- [Hits:1 Misses:5]
Access: 7  HIT  Left  ->    [7, 4, 5] <- Right Replaced:- [Hits:2 Misses:5]
Access: 3  MISS Left  ->    [4, 5, 3] <- Right Replaced:7 [Hits:2 Misses:6]
Access: 4  HIT  Left  ->    [4, 5, 3] <- Right Replaced:- [Hits:3 Misses:6]
Access: 5  HIT  Left  ->    [4, 5, 3] <- Right Replaced:- [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00
```

结果和我们预测的一致

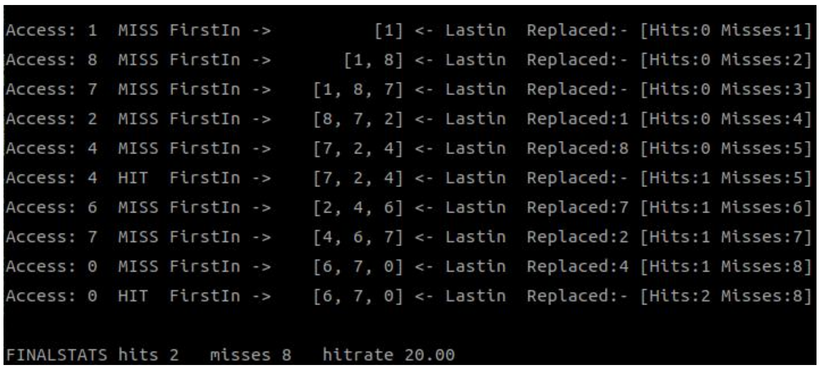
种子 1：
（1）FIFO
python3 paging-policy.py -s 1 -n 10 -p FIFO



```
zlz@zlz-virtual-machine:~$ python3 paging-policy.py -s 1 -n 10 -p FIFO
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 1
ARG notrace False

Assuming a replacement policy of FIFO, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 1  Hit/Miss?  State of Memory?
Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 2  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
```

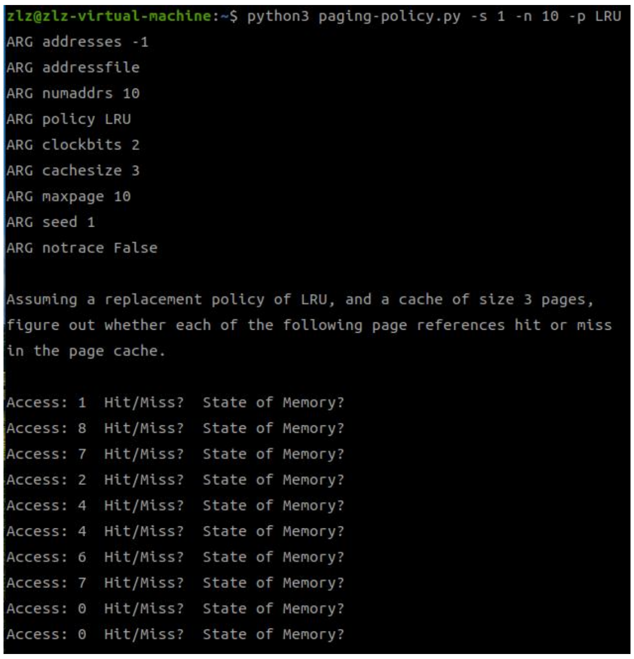
```
Access 1:Miss  [1]     Lastin  Replaced:- [Hits:0 Misses:1]
Access 8:Miss  [1,8]   Lastin  Replaced:- [Hits:0 Misses:2]
Access 7:Miss  [1,8,7] Lastin  Replaced:- [Hits:0 Misses:3]
Access 2:Miss  [8,7,2] Lastin  Replaced:1 [Hits:0 Misses:4]
Access 4:Miss  [7,2,4] Lastin  Replaced:8 [Hits:0 Misses:5]
Access 4:Hit   [7,2,4] Lastin  Replaced:- [Hits:1 Misses:5]
Access 6:Miss  [2,4,6] Lastin  Replaced:7 [Hits:1 Misses:6]
Access 7:Miss  [4,6,7] Lastin  Replaced:2 [Hits:1 Misses:7]
Access 0:Miss  [6,7,0] Lastin  Replaced:4 [Hits:1 Misses:8]
Access 0:Hit   [6,7,0] Lastin  Replaced:- [Hits:2 Misses:8]
```

命中率为 20%



```
Access: 1  MISS FirstIn ->          [1] <- Lastin  Replaced:- [Hits:0 Misses:1]
Access: 8  MISS FirstIn ->       [1, 8] <- Lastin  Replaced:- [Hits:0 Misses:2]
Access: 7  MISS FirstIn ->    [1, 8, 7] <- Lastin  Replaced:- [Hits:0 Misses:3]
Access: 2  MISS FirstIn ->    [8, 7, 2] <- Lastin  Replaced:1 [Hits:0 Misses:4]
Access: 4  MISS FirstIn ->    [7, 2, 4] <- Lastin  Replaced:8 [Hits:0 Misses:5]
Access: 4  HIT  FirstIn ->    [7, 2, 4] <- Lastin  Replaced:- [Hits:1 Misses:5]
Access: 6  MISS FirstIn ->    [2, 4, 6] <- Lastin  Replaced:7 [Hits:1 Misses:6]
Access: 7  MISS FirstIn ->    [4, 6, 7] <- Lastin  Replaced:2 [Hits:1 Misses:7]
Access: 0  MISS FirstIn ->    [6, 7, 0] <- Lastin  Replaced:4 [Hits:1 Misses:8]
Access: 0  HIT  FirstIn ->    [6, 7, 0] <- Lastin  Replaced:- [Hits:2 Misses:8]

FINALSTATS hits 2   misses 8   hitrate 20.00
```

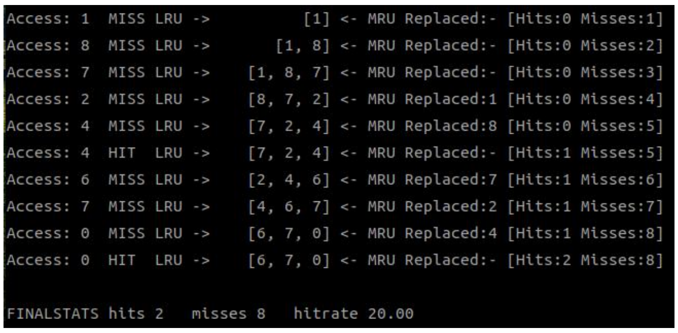
结果和我们预测的一致
（2）LRU
python3 paging-policy.py -s 1 -n 10 -p LRU



```
zlz@zlz-virtual-machine:~$ python3 paging-policy.py -s 1 -n 10 -p LRU
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 1
ARG notrace False

Assuming a replacement policy of LRU, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 1  Hit/Miss?  State of Memory?
Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 2  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
```

```
Access 1:Miss  [1]     Lastin  Replaced:- [Hits:0 Misses:1]
Access 8:Miss  [1,8]   Lastin  Replaced:- [Hits:0 Misses:2]
Access 7:Miss  [1,8,7] Lastin  Replaced:- [Hits:0 Misses:3]
Access 2:Miss  [8,7,2] Lastin  Replaced:1 [Hits:0 Misses:4]
Access 4:Miss  [7,2,4] Lastin  Replaced:8 [Hits:0 Misses:5]
Access 4:Hit   [7,2,4] Lastin  Replaced:- [Hits:1 Misses:5]
Access 6:Miss  [2,4,6] Lastin  Replaced:7 [Hits:1 Misses:6]
Access 7:Miss  [4,6,7] Lastin  Replaced:2 [Hits:1 Misses:7]
Access 0:Miss  [6,7,0] Lastin  Replaced:4 [Hits:1 Misses:8]
Access 0:Hit   [6,7,0] Lastin  Replaced:- [Hits:2 Misses:8]
```

命中率为 20%

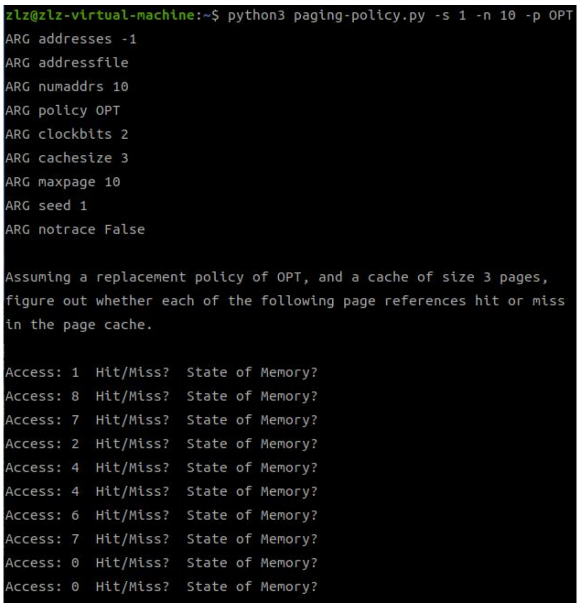


```
Access: 1  MISS LRU ->        [1] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 8  MISS LRU ->     [1, 8] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 7  MISS LRU ->  [1, 8, 7] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 2  MISS LRU ->  [8, 7, 2] <- MRU Replaced:1 [Hits:0 Misses:4]
Access: 4  MISS LRU ->  [7, 2, 4] <- MRU Replaced:8 [Hits:0 Misses:5]
Access: 4  HIT  LRU ->  [7, 2, 4] <- MRU Replaced:- [Hits:1 Misses:5]
Access: 6  MISS LRU ->  [2, 4, 6] <- MRU Replaced:7 [Hits:1 Misses:6]
Access: 7  MISS LRU ->  [4, 6, 7] <- MRU Replaced:2 [Hits:1 Misses:7]
Access: 0  MISS LRU ->  [6, 7, 0] <- MRU Replaced:4 [Hits:1 Misses:8]
Access: 0  HIT  LRU ->  [6, 7, 0] <- MRU Replaced:- [Hits:2 Misses:8]

FINALSTATS hits 2   misses 8   hitrate 20.00
```

结果和我们预测的一致

（3）OPT

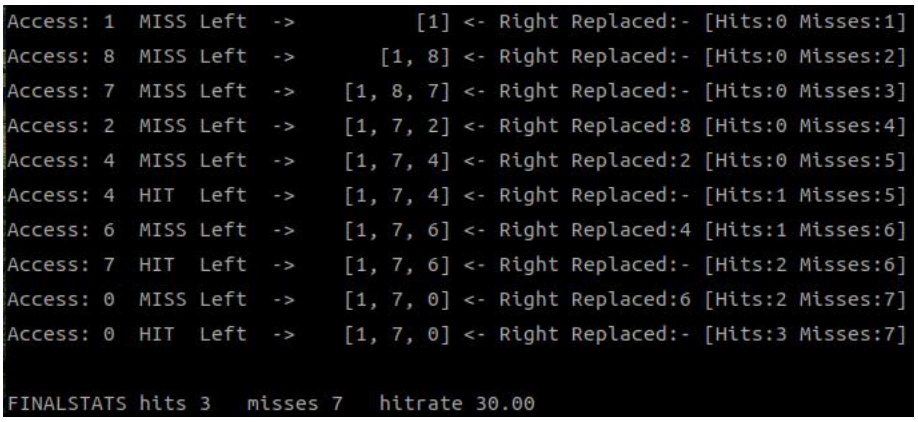
```
python3 paging-policy.py -s 1 -n 10 -p OPT
```



```
zlz@zlz-virtual-machine:~$ python3 paging-policy.py -s 1 -n 10 -p OPT
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy OPT
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 1
ARG notrace False

Assuming a replacement policy of OPT, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 1  Hit/Miss?  State of Memory?
Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 2  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
```

```
Access 1:Miss  [1]     Lastin  Replaced:- [Hits:0 Misses:1]
Access 8:Miss  [1,8]   Lastin  Replaced:- [Hits:0 Misses:2]
Access 7:Miss  [1,8,7] Lastin  Replaced:- [Hits:0 Misses:3]
Access 2:Miss  [8,7,2] Lastin  Replaced:1 [Hits:0 Misses:4]
Access 4:Miss  [7,2,4] Lastin  Replaced:8 [Hits:0 Misses:5]
Access 4:Hit   [7,2,4] Lastin  Replaced:- [Hits:1 Misses:5]
Access 6:Miss  [7,4,6] Lastin  Replaced:2 [Hits:1 Misses:6]
Access 7:Hit   [7,4,6] Lastin  Replaced:- [Hits:2 Misses:6]
Access 0:Miss  [4,6,0] Lastin  Replaced:7 [Hits:2 Misses:7]
Access 0:Hit   [4,6,0] Lastin  Replaced:- [Hits:3 Misses:7]
```

命中率为 30%



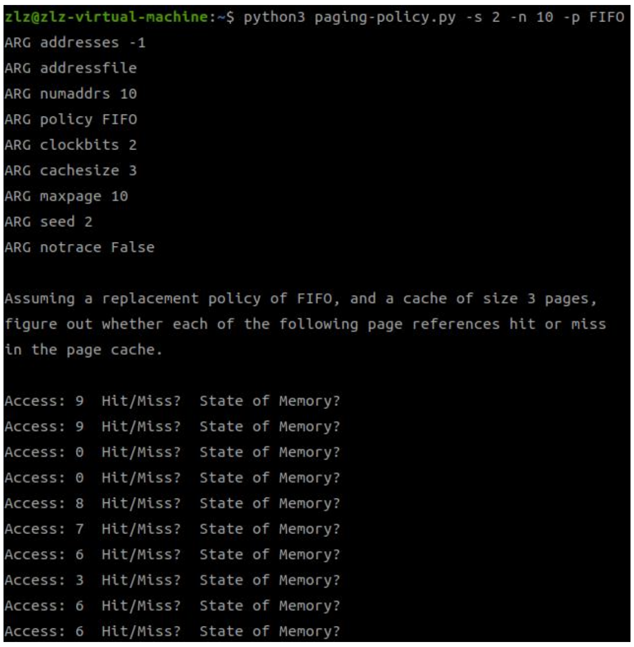
```
Access: 1  MISS Left  ->          [1] <- Right Replaced:- [Hits:0 Misses:1]
Access: 8  MISS Left  ->       [1, 8] <- Right Replaced:- [Hits:0 Misses:2]
Access: 7  MISS Left  ->    [1, 8, 7] <- Right Replaced:- [Hits:0 Misses:3]
Access: 2  MISS Left  ->    [1, 7, 2] <- Right Replaced:8 [Hits:0 Misses:4]
Access: 4  MISS Left  ->    [1, 7, 4] <- Right Replaced:2 [Hits:0 Misses:5]
Access: 4  HIT  Left  ->    [1, 7, 4] <- Right Replaced:- [Hits:1 Misses:5]
Access: 6  MISS Left  ->    [1, 7, 6] <- Right Replaced:4 [Hits:1 Misses:6]
Access: 7  HIT  Left  ->    [1, 7, 6] <- Right Replaced:- [Hits:2 Misses:6]
Access: 0  MISS Left  ->    [1, 7, 0] <- Right Replaced:6 [Hits:2 Misses:7]
Access: 0  HIT  Left  ->    [1, 7, 0] <- Right Replaced:- [Hits:3 Misses:7]

FINALSTATS hits 3   misses 7   hitrate 30.00
```

结果和我们预测的一致

种子 2：

（1）FIFO

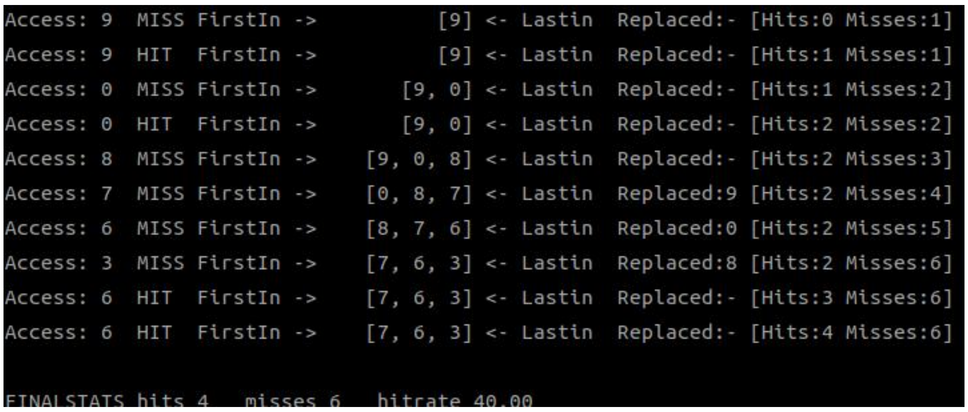
python3 paging-policy.py -s 2 -n 10 -p FIFO



```
zlz@zlz-virtual-machine:~$ python3 paging-policy.py -s 2 -n 10 -p FIFO
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 2
ARG notrace False

Assuming a replacement policy of FIFO, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 9  Hit/Miss?  State of Memory?
Access: 9  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 3  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
```

```
Access 9:Miss  [9]     Lastin  Replaced:- [Hits:0 Misses:1]
Access 9:Hit   [9]     Lastin  Replaced:- [Hits:1 Misses:1]
Access 0:Miss  [9,0]   Lastin  Replaced:- [Hits:1 Misses:2]
Access 0:Hit   [9,0]   Lastin  Replaced:- [Hits:2 Misses:2]
Access 8:Miss  [9,0,8] Lastin  Replaced:- [Hits:2 Misses:3]
Access 7:Miss  [0,8,7] Lastin  Replaced:9 [Hits:2 Misses:4]
Access 6:Miss  [8,7,6] Lastin  Replaced:0 [Hits:2 Misses:5]
Access 3:Miss  [7,6,3] Lastin  Replaced:8 [Hits:2 Misses:6]
Access 6:Hit   [7,6,3] Lastin  Replaced:- [Hits:3 Misses:6]
Access 6:Hit   [7,6,3] Lastin  Replaced:- [Hits:4 Misses:6]
```

命中率为 40%

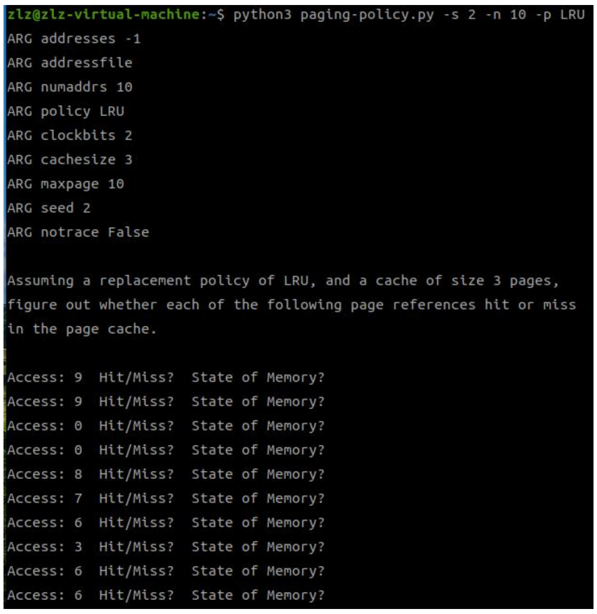


```
Access: 9  MISS FirstIn ->          [9] <- Lastin  Replaced:- [Hits:0 Misses:1]
Access: 9  HIT  FirstIn ->          [9] <- Lastin  Replaced:- [Hits:1 Misses:1]
Access: 0  MISS FirstIn ->       [9, 0] <- Lastin  Replaced:- [Hits:1 Misses:2]
Access: 0  HIT  FirstIn ->       [9, 0] <- Lastin  Replaced:- [Hits:2 Misses:2]
Access: 8  MISS FirstIn ->    [9, 0, 8] <- Lastin  Replaced:- [Hits:2 Misses:3]
Access: 7  MISS FirstIn ->    [0, 8, 7] <- Lastin  Replaced:9 [Hits:2 Misses:4]
Access: 6  MISS FirstIn ->    [8, 7, 6] <- Lastin  Replaced:0 [Hits:2 Misses:5]
Access: 3  MISS FirstIn ->    [7, 6, 3] <- Lastin  Replaced:8 [Hits:2 Misses:6]
Access: 6  HIT  FirstIn ->    [7, 6, 3] <- Lastin  Replaced:- [Hits:3 Misses:6]
Access: 6  HIT  FirstIn ->    [7, 6, 3] <- Lastin  Replaced:- [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00
```

结果和我们预测的一致

（2）LRU

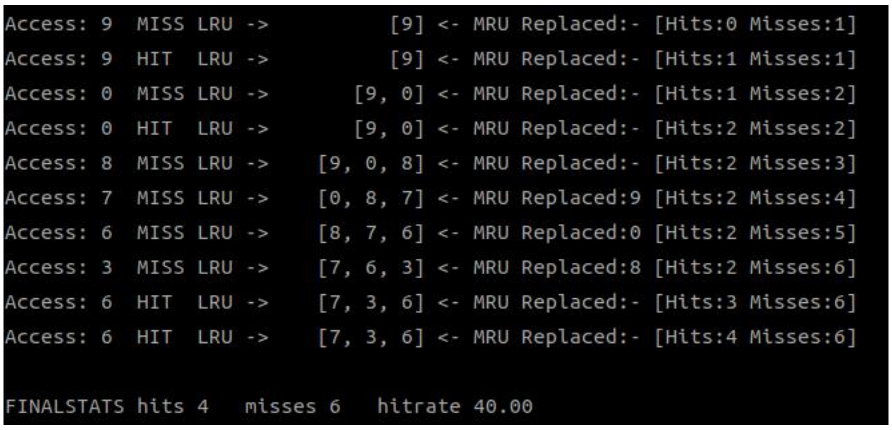
python3 paging-policy.py -s 2 -n 10 -p LRU



```
zlz@zlz-virtual-machine:~$ python3 paging-policy.py -s 2 -n 10 -p LRU
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 2
ARG notrace False

Assuming a replacement policy of LRU, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 9  Hit/Miss?  State of Memory?
Access: 9  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 3  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
```

```
Access 9:Miss  [9]     Lastin  Replaced:- [Hits:0 Misses:1]
Access 9:Hit   [9]     Lastin  Replaced:- [Hits:1 Misses:1]
Access 0:Miss  [9,0]   Lastin  Replaced:- [Hits:1 Misses:2]
Access 0:Hit   [9,0]   Lastin  Replaced:- [Hits:2 Misses:2]
Access 8:Miss  [9,0,8] Lastin  Replaced:- [Hits:2 Misses:3]
Access 7:Miss  [0,8,7] Lastin  Replaced:9 [Hits:2 Misses:4]
Access 6:Miss  [8,7,6] Lastin  Replaced:0 [Hits:2 Misses:5]
Access 3:Miss  [7,3,6] Lastin  Replaced:8 [Hits:2 Misses:6]
Access 6:Hit   [7,3,6] Lastin  Replaced:- [Hits:3 Misses:6]
Access 6:Hit   [7,3,6] Lastin  Replaced:- [Hits:4 Misses:6]
```

命中率为 40%

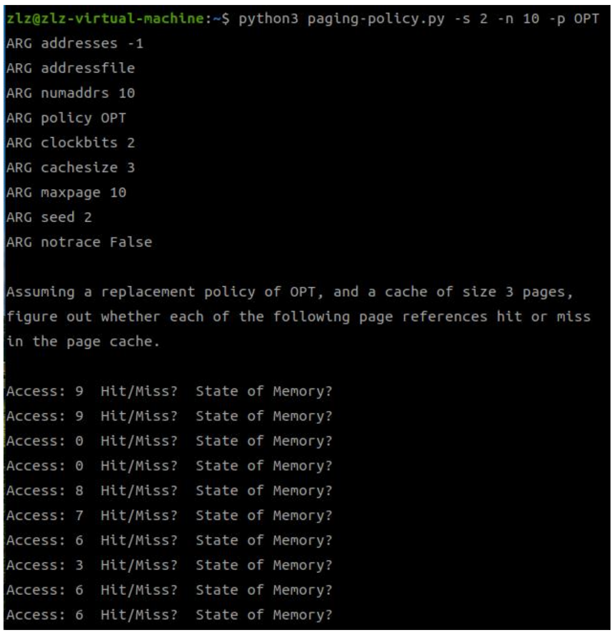


```
Access: 9  MISS LRU ->          [9] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 9  HIT  LRU ->          [9] <- MRU Replaced:- [Hits:1 Misses:1]
Access: 0  MISS LRU ->       [9, 0] <- MRU Replaced:- [Hits:1 Misses:2]
Access: 0  HIT  LRU ->       [9, 0] <- MRU Replaced:- [Hits:2 Misses:2]
Access: 8  MISS LRU ->    [9, 0, 8] <- MRU Replaced:- [Hits:2 Misses:3]
Access: 7  MISS LRU ->    [0, 8, 7] <- MRU Replaced:9 [Hits:2 Misses:4]
Access: 6  MISS LRU ->    [8, 7, 6] <- MRU Replaced:0 [Hits:2 Misses:5]
Access: 3  MISS LRU ->    [7, 6, 3] <- MRU Replaced:8 [Hits:2 Misses:6]
Access: 6  HIT  LRU ->    [7, 3, 6] <- MRU Replaced:- [Hits:3 Misses:6]
Access: 6  HIT  LRU ->    [7, 3, 6] <- MRU Replaced:- [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00
```

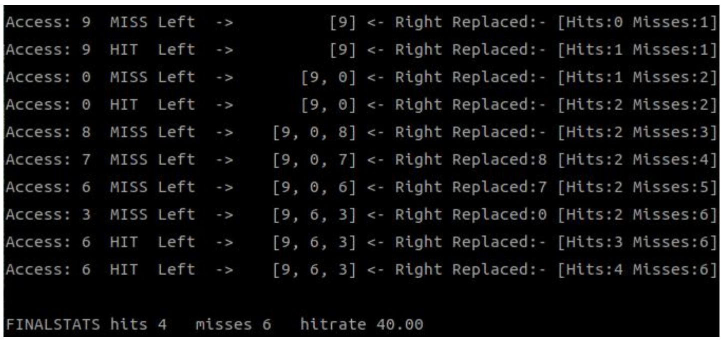
结果和我们预测的一致

（3）OPT

python3 paging-policy.py -s 2 -n 10 -p OPT


```
zlz@zlz-virtual-machine:~$ python3 paging-policy.py -s 2 -n 10 -p OPT
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy OPT
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 2
ARG notrace False

Assuming a replacement policy of OPT, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 9  Hit/Miss?  State of Memory?
Access: 9  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 3  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
```

```
Access 9:Miss  [9]     Lastin  Replaced:- [Hits:0 Misses:1]
Access 9:Hit   [9]     Lastin  Replaced:- [Hits:1 Misses:1]
Access 0:Miss  [9,0]   Lastin  Replaced:- [Hits:1 Misses:2]
Access 0:Hit   [9,0]   Lastin  Replaced:- [Hits:2 Misses:2]
Access 8:Miss  [9,0,8] Lastin  Replaced:- [Hits:2 Misses:3]
Access 7:Miss  [9,0,7] Lastin  Replaced:8 [Hits:2 Misses:4]
Access 6:Miss  [9,7,6] Lastin  Replaced:7 [Hits:2 Misses:5]
Access 3:Miss  [9,6,3] Lastin  Replaced:0 [Hits:2 Misses:6]
Access 6:Hit   [9,6,3] Lastin  Replaced:- [Hits:3 Misses:6]
Access 6:Hit   [9,6,3] Lastin  Replaced:- [Hits:4 Misses:6]
```
命中率为 40%


```
Access: 9  MISS Left  ->          [9] <- Right Replaced:- [Hits:0 Misses:1]
Access: 9  HIT  Left  ->          [9] <- Right Replaced:- [Hits:1 Misses:1]
Access: 0  MISS Left  ->       [9, 0] <- Right Replaced:- [Hits:1 Misses:2]
Access: 0  HIT  Left  ->       [9, 0] <- Right Replaced:- [Hits:2 Misses:2]
Access: 8  MISS Left  ->    [9, 0, 8] <- Right Replaced:- [Hits:2 Misses:3]
Access: 7  MISS Left  ->    [9, 0, 7] <- Right Replaced:8 [Hits:2 Misses:4]
Access: 6  MISS Left  ->    [9, 0, 6] <- Right Replaced:7 [Hits:2 Misses:5]
Access: 3  MISS Left  ->    [9, 6, 3] <- Right Replaced:0 [Hits:2 Misses:6]
Access: 6  HIT  Left  ->    [9, 6, 3] <- Right Replaced:- [Hits:3 Misses:6]
Access: 6  HIT  Left  ->    [9, 6, 3] <- Right Replaced:- [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00
```
结果和我们预测的一致

2.对于大小为 5 的高速缓存，为以下每个策略生成最差情况的地址引用序列：FIFO、LRU 和 MRU（最差情况下的引用序列导致尽可能多的未命中）。对于最差情况下的引用序列，需要的缓存增大多少，才能大幅提高性能，并接近 OPT?

FIFO:1,2,3,4,5,6

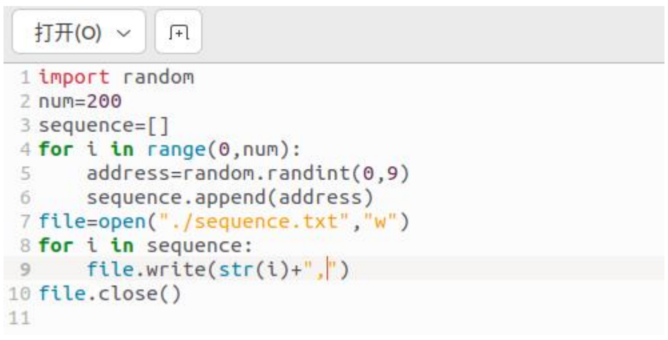
LRU:1,2,3,4,5,6

MRU:1,2,3,4,5,6,5,6,5,6

对于 FIFO 和 LRU，只要缓存大小和序列大小相同就可以提高命中率，除了冷不

命中其他都命中。
对于 MRU，如果缓存容量刚好满，而且有两页交替访问，就会出现抖动，这种情况下可以适当提高缓存大小，比如缓存增大 1，就可以提高命中率。

3.生成一个随机追踪序列（使用 Python 或 Perl）。你预计不同的策略在这样的追踪序列上的表现如何?



```python
import random
num=200
sequence=[]
for i in range(0,num):
    address=random.randint(0,9)
    sequence.append(address)
file=open("./sequence.txt","w")
for i in sequence:
    file.write(str(i)+",")
file.close()
```

生成的序列如下：
8,1,8,5,9,3,7,6,6,1,0,5,2,3,5,0,3,7,3,9,2,6,4,0,3,3,2,5,9,6,1,0,2,2,5,0,5,2,7,2,7,5,2,8,2,9,0,3,1,5,9,8,6,6,5,7,0,2,6,7,0,3,7,0,7,5,2,9,0,5,3,9,2,6,7,8,3,6,9,3,1,0,9,1,4,0,4,3,4,6,1,4,0,6,0,7,6,5,5,3,4,8,7,8,3,1,7,5,3,7,1,1,2,9,4,8,6,1,1,8,8,9,8,9,7,1,6,4,1,3,4,4,3,4,8,7,3,8,5,8,5,1,1,0,3,7,6,5,8,8,0,1,7,8,1,4,9,7,8,1,1,5,1,2,2,0,1,8,4,3,9,2,8,0,0,7,0,9,6,2,1,1,5,8,2,9,9,2,1,3,2,9,0,2,3,8,4,1,2,2

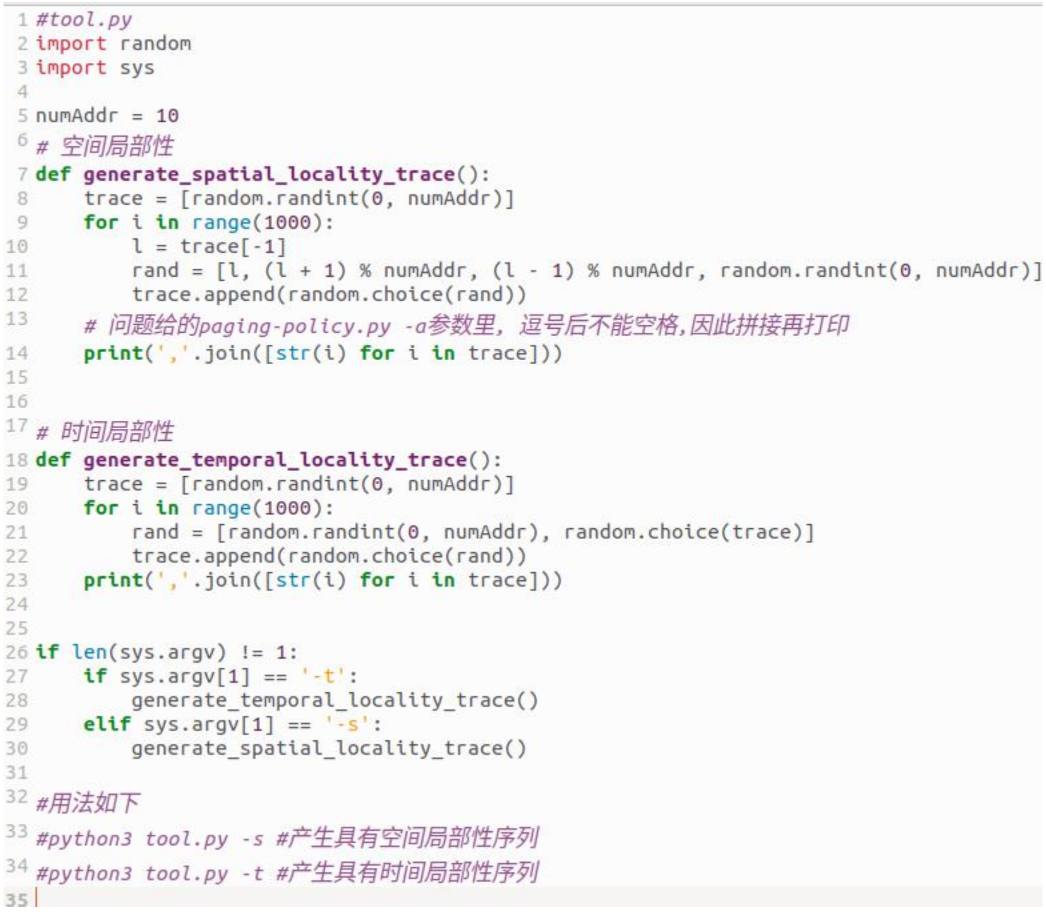
使用下面命令：

```
python3 paging-policy.py -n 10 -p OPT -c -a （序列）
python3 paging-policy.py -n 10 -p FIFO -c -a （序列）
python3 paging-policy.py -n 10 -p LRU -c -a （序列）
python3 paging-policy.py -n 10 -p CLOCK -c -a （序列）
python3 paging-policy.py -n 10 -p RAND -c -a （序列）
python3 paging-policy.py -n 10 -p MRU -c -a （序列）
python3 paging-policy.py -n 10 -p UNOPT -c -a （序列）
```

策略	命中率
OPT	51.50%
FIFO	32.00%
LRU	31.50%
CLOCK	28.00%
RAND	32.50%
MRU	26.50%
UNOPT	10.50%

在这种情况下没有局部性，随机进行访问，预计除 OPT 外的策略效果基本相近。结果符合预期。

4.现在生成一些局部性追踪序列。如何能够产生这样的追踪序列？LRU 表现如何？RAND 比 LRU 好多少？CLOCK 表现如何？CLOCK 使用不同数量的时钟位，表现如何？



```python
#tool.py
import random
import sys

numAddr = 10
# 空间局部性
def generate_spatial_locality_trace():
    trace = [random.randint(0, numAddr)]
    for i in range(1000):
        l = trace[-1]
        rand = [l, (l + 1) % numAddr, (l - 1) % numAddr, random.randint(0, numAddr)]
        trace.append(random.choice(rand))
    # 问题给的paging-policy.py -a参数里，逗号后不能空格,因此拼接再打印
    print(','.join([str(i) for i in trace]))


# 时间局部性
def generate_temporal_locality_trace():
    trace = [random.randint(0, numAddr)]
    for i in range(1000):
        rand = [random.randint(0, numAddr), random.choice(trace)]
        trace.append(random.choice(rand))
    print(','.join([str(i) for i in trace]))


if len(sys.argv) != 1:
    if sys.argv[1] == '-t':
        generate_temporal_locality_trace()
    elif sys.argv[1] == '-s':
        generate_spatial_locality_trace()

#用法如下
#python3 tool.py -s #产生具有空间局部性序列
#python3 tool.py -t #产生具有时间局部性序列

```

使用下面命令进行测试：

#测试空间局部性

```
python3 paging-policy.py -c -N -a  $(python3 2.py -s) -p LRU
python3 paging-policy.py -c -N -a  $(python3 2.py -s) -p RAND
python3 paging-policy.py -c -N -a  $(python3 2.py -s) -p CLOCK
python3 paging-policy.py -c -N -a  $(python3 2.py -s) -p OPT
```

#测试时间局部性

```
python3 paging-policy.py -c -N -a  $(python3 2.py -s) -p LRU
python3 paging-policy.py -c -N -a  $(python3 2.py -s) -p RAND
python3 paging-policy.py -c -N -a  $(python3 2.py -s) -p CLOCK
python3 paging-policy.py -c -N -a  $(python3 2.py -s) -p OPT
```

测试结果如下：

策略	时间局部性-命中率	空间局部性-命中率
LRU	29.87%	55.44%
RAND	30.57%	53.35%
CLOCK	30.47%	52.95%

策略	时间局部性-命中率	空间局部性-命中率
OPT	51.75%	66.33%

可见，OPT 效果最好。
LRU 在空间局部性好的序列上效果较好。CLOCK，RAND 和 LRU 表现差不多，但比 OPT 差距相对较大。

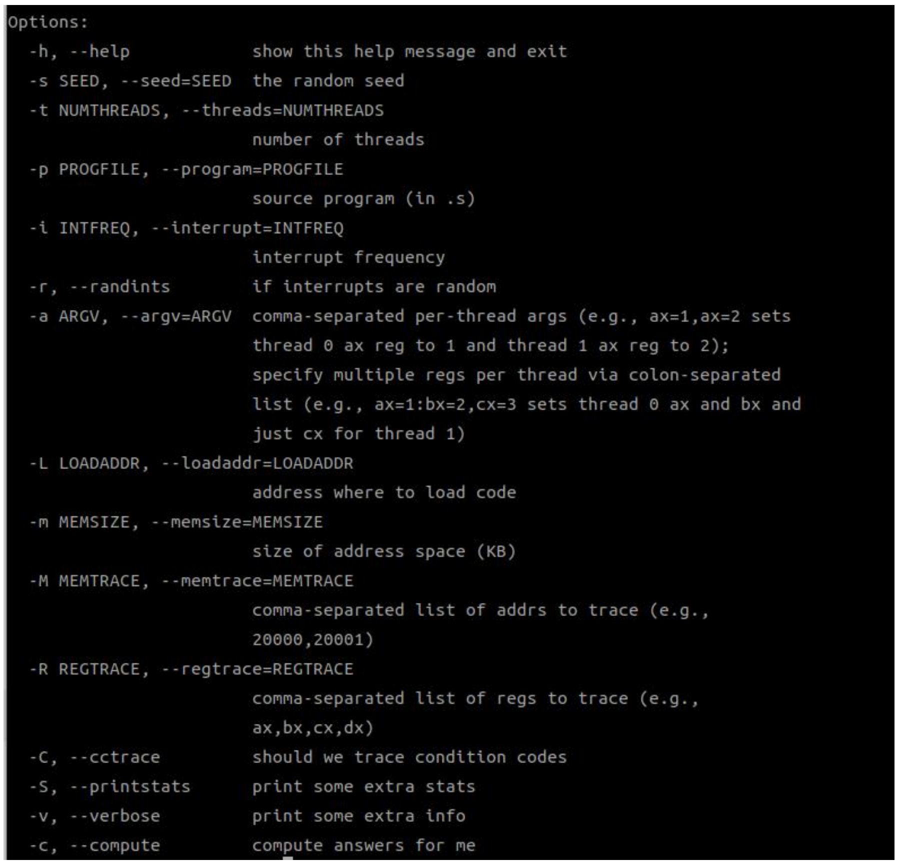
对于 CLOCK 策略，使用如下指令改变时钟位
```
python3 paging-policy.py -c -N -a $(python3 2.py -s) -p CLOCK -b 1
python3 paging-policy.py -c -N -a $(python3 2.py -s) -p CLOCK -b 2
python3 paging-policy.py -c -N -a $(python3 2.py -s) -p CLOCK -b 3
python3 paging-policy.py -c -N -a $(python3 2.py -s) -p CLOCK -b 4
python3 paging-policy.py -c -N -a $(python3 2.py -s) -p CLOCK -b 5
python3 paging-policy.py -c -N -a $(python3 2.py -s) -p CLOCK -b 6
python3 paging-policy.py -c -N -a $(python3 2.py -s) -p CLOCK -b 7
python3 paging-policy.py -c -N -a $(python3 2.py -s) -p CLOCK -b 8
python3 paging-policy.py -c -N -a $(python3 2.py -s) -p CLOCK -b 9
python3 paging-policy.py -c -N -a $(python3 2.py -s) -p CLOCK -b 10
```
测试结果如下：

时钟位	时间局部性-命中率	空间局部性-命中率
1	30.27%	51.35%
2	31.67%	52.95%
3	30.37%	52.05%
4	31.67%	53.25%
5	32.17%	51.05%
6	31.27%	50.75%
7	31.37%	51.15%
8	31.47%	51.45%
9	31.47%	50.55%
10	31.47%	50.55%

可以观察到，在一定的范围内，随着时钟位的增加，CLOCK 策略的效果逐步提升。
但是超过一定范围效果会减弱直至保持不变

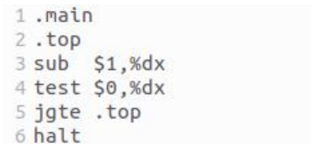
第 26 章

1.开始,我们来看一个简单的程序,“loop.s”。首先,阅读这个程序,看看你是否能理解它: cat loop.s。然后,用这些参数运行它:./x86.py -p loop.s -t 1 -i 100 -R dx
这指定了一个单线程,每 100 条指令产生一个中断,并且追踪寄存器 %d。你能弄清楚 %dx 在运行过程中的值吗? 你有答案之后,运行上面的代码并使用 -c 标志来检查你的答案。注意答案的左边显示了右侧指令运行后寄存器的值(或内存的值)



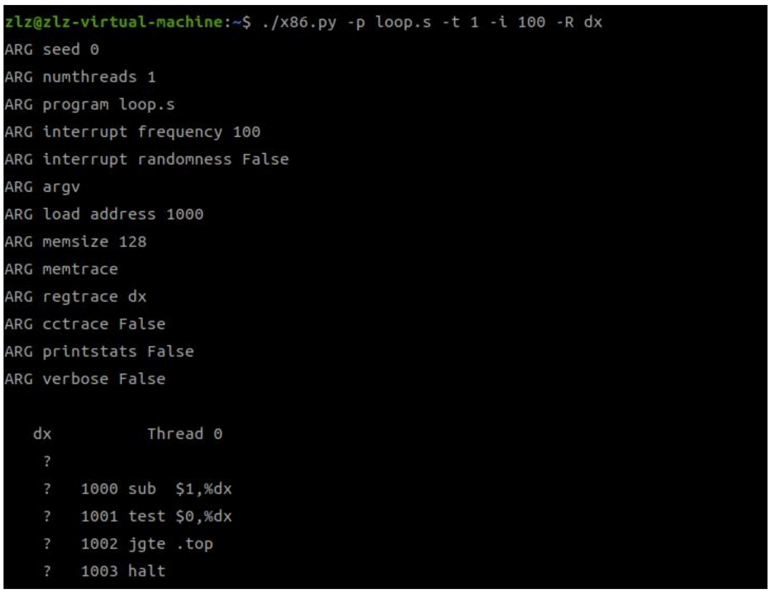
```
Options:
  -h, --help            show this help message and exit
  -s SEED, --seed=SEED  the random seed
  -t NUMTHREADS, --threads=NUMTHREADS
                        number of threads
  -p PROGFILE, --program=PROGFILE
                        source program (in .s)
  -i INTFREQ, --interrupt=INTFREQ
                        interrupt frequency
  -r, --randints        if interrupts are random
  -a ARGV, --argv=ARGV  comma-separated per-thread args (e.g., ax=1,ax=2 sets
                        thread 0 ax reg to 1 and thread 1 ax reg to 2);
                        specify multiple regs per thread via colon-separated
                        list (e.g., ax=1:bx=2,cx=3 sets thread 0 ax and bx and
                        just cx for thread 1)
  -L LOADADDR, --loadaddr=LOADADDR
                        address where to load code
  -m MEMSIZE, --memsize=MEMSIZE
                        size of address space (KB)
  -M MEMTRACE, --memtrace=MEMTRACE
                        comma-separated list of addrs to trace (e.g.,
                        20000,20001)
  -R REGTRACE, --regtrace=REGTRACE
                        comma-separated list of regs to trace (e.g.,
                        ax,bx,cx,dx)
  -C, --cctrace         should we trace condition codes
  -S, --printstats      print some extra stats
  -v, --verbose         print some extra info
  -c, --compute         compute answers for me
```

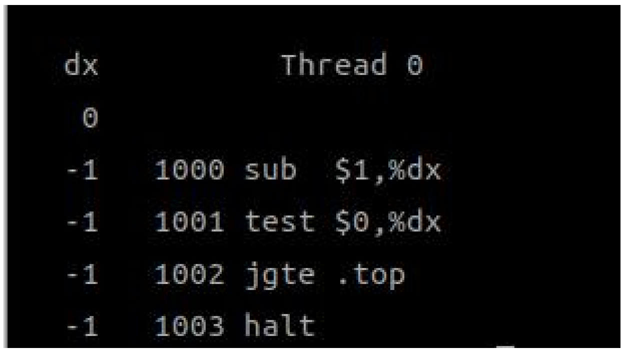
首先，我们可以先查看 loop.s 文件



```
.main
.top
sub  $1,%dx
test $0,%dx
jgte .top
halt
```

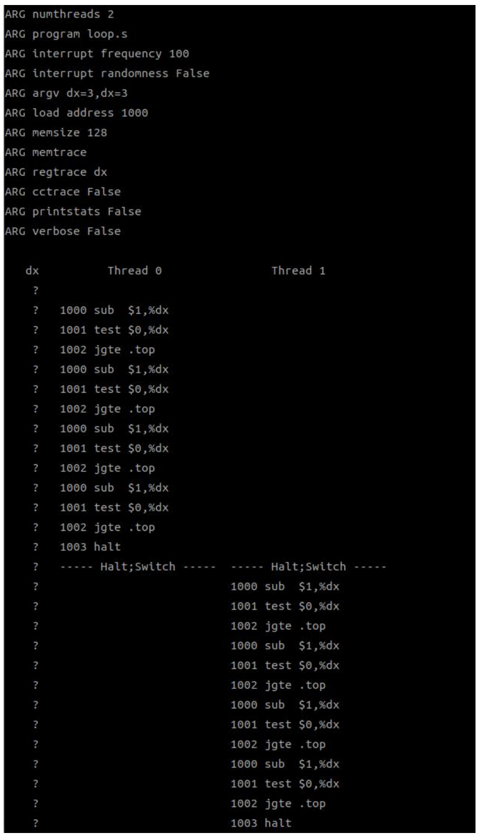
相当于将 dx 寄存器中的值-1，然后和 0 取与，如果值大于等于 0，就跳转到.top 处


```
zlz@zlz-virtual-machine:~$ ./x86.py -p loop.s -t 1 -i 100 -R dx
ARG seed 0
ARG numthreads 1
ARG program loop.s
ARG interrupt frequency 100
ARG interrupt randomness False
ARG argv
ARG load address 1000
ARG memsize 128
ARG memtrace
ARG regtrace dx
ARG cctrace False
ARG printstats False
ARG verbose False

   dx          Thread 0
    ?
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1003 halt
```
只执行了 4 条指令就结束了，那么 dx 的初始值应该为 0，进入循环后为-1，直接退出了循环。


```
   dx          Thread 0
    0
   -1   1000 sub  $1,%dx
   -1   1001 test $0,%dx
   -1   1002 jgte .top
   -1   1003 halt
```
用-c 查看 dx 的值，结果符合我们的预期

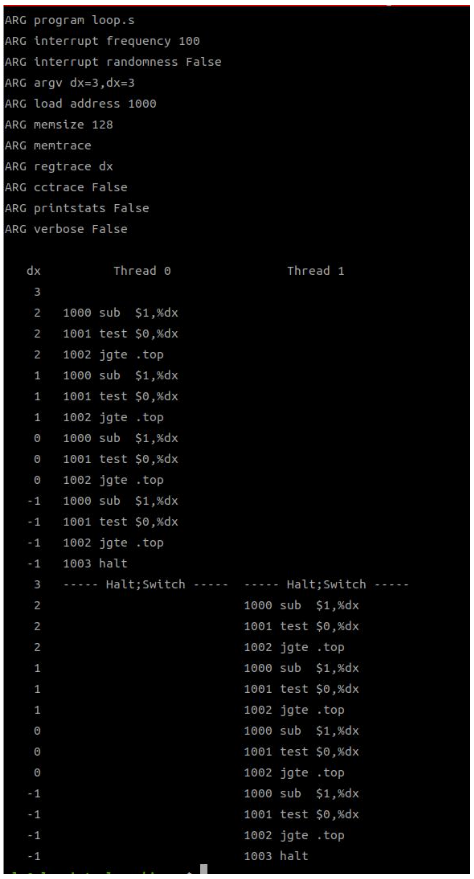
2.现在运行相同的代码,但使用这些标志:
/x86.py -p loop.s -t 2 -i 100 -a dx=3,dx=3 -R dx
这指定了两个线程,并将每个%dx 寄存器初始化为 3。%dx 会看到什么值?使用-c 标志运行以查看答案。多个线程的存在是否会影响计算？这段代码有竞态条件吗?

多个线程的存在不会影响计算，因为代码的执行时长小于中断周期，代码段可以



```
ARG numthreads 2
ARG program loop.s
ARG interrupt frequency 100
ARG interrupt randomness False
ARG argv dx=3,dx=3
ARG load address 1000
ARG memsize 128
ARG memtrace
ARG regtrace dx
ARG cctrace False
ARG printstats False
ARG verbose False

   dx          Thread 0                Thread 1
    ?
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1003 halt
    ?   ----- Halt;Switch -----  ----- Halt;Switch -----
    ?                            1000 sub  $1,%dx
    ?                            1001 test $0,%dx
    ?                            1002 jgte .top
    ?                            1000 sub  $1,%dx
    ?                            1001 test $0,%dx
    ?                            1002 jgte .top
    ?                            1000 sub  $1,%dx
    ?                            1001 test $0,%dx
    ?                            1002 jgte .top
    ?                            1000 sub  $1,%dx
    ?                            1001 test $0,%dx
    ?                            1002 jgte .top
    ?                            1003 halt
```

运行-c 查看答案



```
ARG program loop.s
ARG interrupt frequency 100
ARG interrupt randomness False
ARG argv dx=3,dx=3
ARG load address 1000
ARG memsize 128
ARG memtrace
ARG regtrace dx
ARG cctrace False
ARG printstats False
ARG verbose False

   dx          Thread 0                Thread 1
    3
    2   1000 sub  $1,%dx
    2   1001 test $0,%dx
    2   1002 jgte .top
    1   1000 sub  $1,%dx
    1   1001 test $0,%dx
    1   1002 jgte .top
    0   1000 sub  $1,%dx
    0   1001 test $0,%dx
    0   1002 jgte .top
   -1   1000 sub  $1,%dx
   -1   1001 test $0,%dx
   -1   1002 jgte .top
   -1   1003 halt
    3   ----- Halt;Switch -----  ----- Halt;Switch -----
    2                            1000 sub  $1,%dx
    2                            1001 test $0,%dx
    2                            1002 jgte .top
    1                            1000 sub  $1,%dx
    1                            1001 test $0,%dx
    1                            1002 jgte .top
    0                            1000 sub  $1,%dx
    0                            1001 test $0,%dx
    0                            1002 jgte .top
   -1                            1000 sub  $1,%dx
   -1                            1001 test $0,%dx
   -1                            1002 jgte .top
   -1                            1003 halt
```

多个线程的存在不会影响计算，因为代码的执行时长小于中断周期，代码段可以完全执行完，并且代码本身没有竞争条件，不会到临界区，产生结果错误

这段代码没有竞争条件，因为其中不存在临界区。

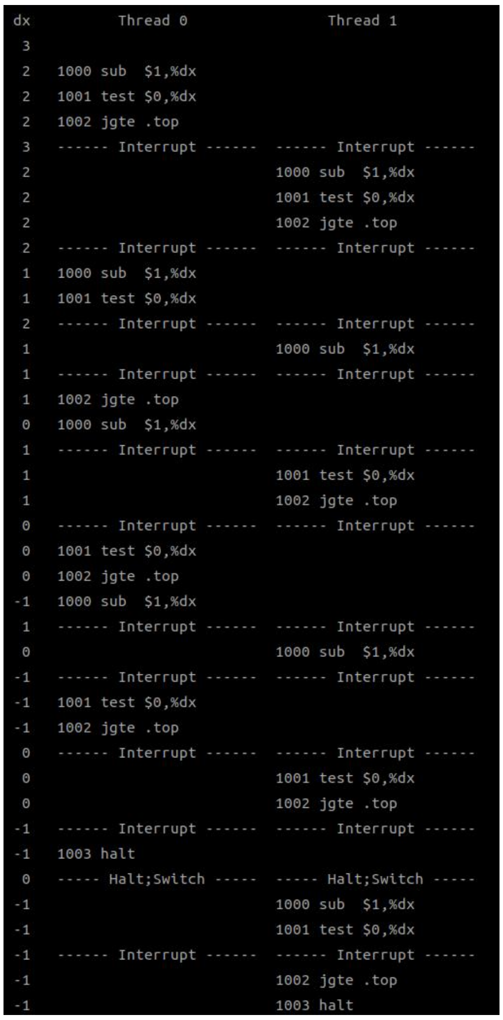
3.现在运行以下命令:

./x86.py -p loop.s -t 2 -i 3 -r -a dx=3,dx=3 -R dx

这使得中断间隔非常小且随机，使用不同的种子和-s 来查看不同的交替、中断频率是否会改变这个程序的行为?

种子 0：

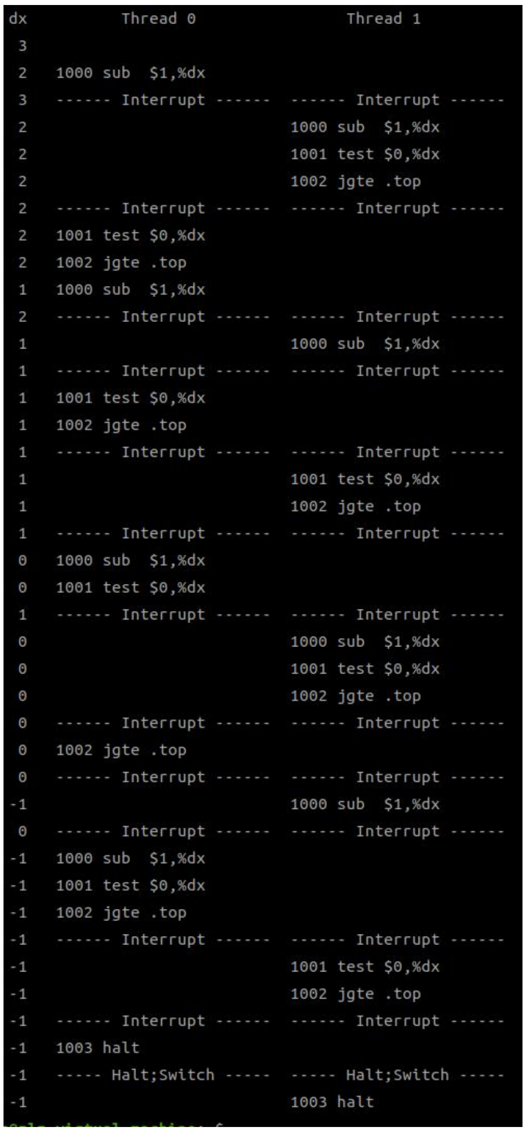
```
./x86.py -p loop.s -t 2 -i 3 -r -a dx=3,dx=3 -R dx -c -s 0
```



```
dx          Thread 0                Thread 1
 3
 2   1000 sub  $1,%dx
 2   1001 test $0,%dx
 2   1002 jgte .top
 3   ------ Interrupt ------  ------ Interrupt ------
 2                            1000 sub  $1,%dx
 2                            1001 test $0,%dx
 2                            1002 jgte .top
 2   ------ Interrupt ------  ------ Interrupt ------
 1   1000 sub  $1,%dx
 1   1001 test $0,%dx
 2   ------ Interrupt ------  ------ Interrupt ------
 1                            1000 sub  $1,%dx
 1   ------ Interrupt ------  ------ Interrupt ------
 1   1002 jgte .top
 0   1000 sub  $1,%dx
 1   ------ Interrupt ------  ------ Interrupt ------
 1                            1001 test $0,%dx
 1                            1002 jgte .top
 0   ------ Interrupt ------  ------ Interrupt ------
 0   1001 test $0,%dx
 0   1002 jgte .top
-1   1000 sub  $1,%dx
 1   ------ Interrupt ------  ------ Interrupt ------
 0                            1000 sub  $1,%dx
-1   ------ Interrupt ------  ------ Interrupt ------
-1   1001 test $0,%dx
-1   1002 jgte .top
 0   ------ Interrupt ------  ------ Interrupt ------
 0                            1001 test $0,%dx
 0                            1002 jgte .top
-1   ------ Interrupt ------  ------ Interrupt ------
-1   1003 halt
 0   ----- Halt;Switch -----  ----- Halt;Switch -----
-1                            1000 sub  $1,%dx
-1                            1001 test $0,%dx
-1   ------ Interrupt ------  ------ Interrupt ------
-1                            1002 jgte .top
-1                            1003 halt
```

种子 1：

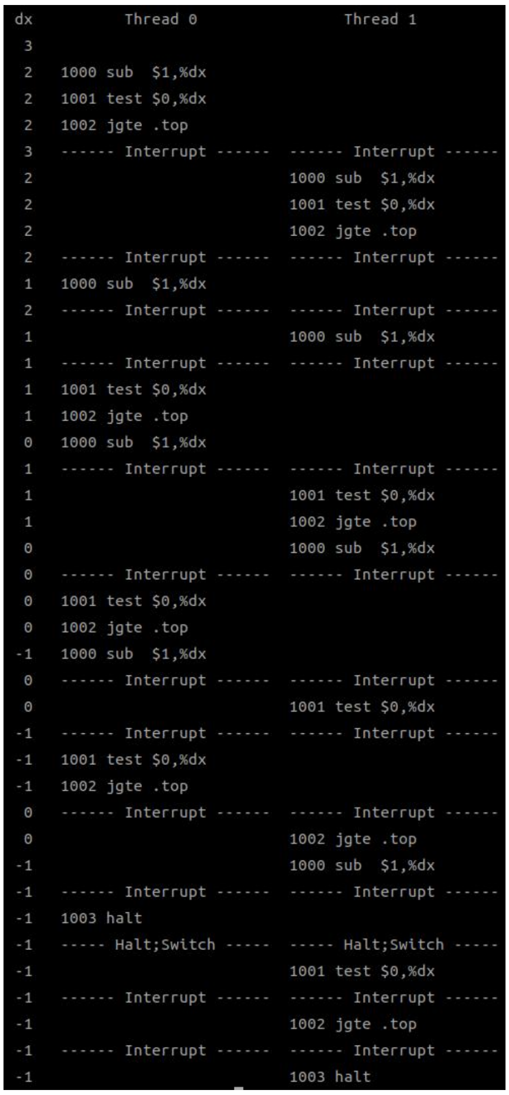
```
./x86.py -p loop.s -t 2 -i 3 -r -a dx=3,dx=3 -R dx -c -s 1
```



```
dx          Thread 0                Thread 1
 3
 2   1000 sub  $1,%dx
 3   ------ Interrupt ------  ------ Interrupt ------
 2                            1000 sub  $1,%dx
 2                            1001 test $0,%dx
 2                            1002 jgte .top
 2   ------ Interrupt ------  ------ Interrupt ------
 2   1001 test $0,%dx
 2   1002 jgte .top
 1   1000 sub  $1,%dx
 2   ------ Interrupt ------  ------ Interrupt ------
 1                            1000 sub  $1,%dx
 1   ------ Interrupt ------  ------ Interrupt ------
 1   1001 test $0,%dx
 1   1002 jgte .top
 1   ------ Interrupt ------  ------ Interrupt ------
 1                            1001 test $0,%dx
 1                            1002 jgte .top
 1   ------ Interrupt ------  ------ Interrupt ------
 0   1000 sub  $1,%dx
 0   1001 test $0,%dx
 1   ------ Interrupt ------  ------ Interrupt ------
 0                            1000 sub  $1,%dx
 0                            1001 test $0,%dx
 0                            1002 jgte .top
 0   ------ Interrupt ------  ------ Interrupt ------
 0   1002 jgte .top
 0   ------ Interrupt ------  ------ Interrupt ------
-1                            1000 sub  $1,%dx
 0   ------ Interrupt ------  ------ Interrupt ------
-1   1000 sub  $1,%dx
-1   1001 test $0,%dx
-1   1002 jgte .top
-1   ------ Interrupt ------  ------ Interrupt ------
-1                            1001 test $0,%dx
-1                            1002 jgte .top
-1   ------ Interrupt ------  ------ Interrupt ------
-1   1003 halt
-1   ----- Halt;Switch -----  ----- Halt;Switch -----
-1                            1003 halt
```

种子 2：

./x86.py -p loop.s -t 2 -i 3 -r -a dx=3,dx=3 -R dx -c -s 2

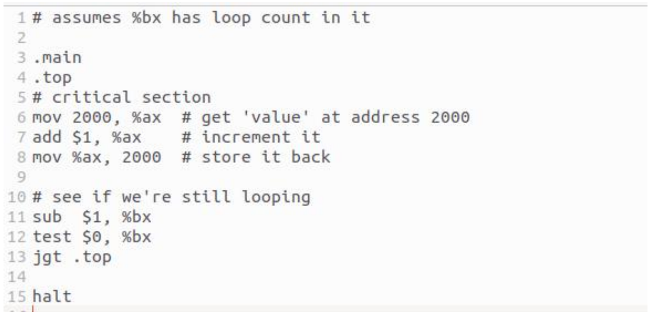


```
dx          Thread 0                Thread 1
 3
 2   1000 sub  $1,%dx
 2   1001 test $0,%dx
 2   1002 jgte .top
 3   ------ Interrupt ------  ------ Interrupt ------
 2                            1000 sub  $1,%dx
 2                            1001 test $0,%dx
 2                            1002 jgte .top
 2   ------ Interrupt ------  ------ Interrupt ------
 1   1000 sub  $1,%dx
 2   ------ Interrupt ------  ------ Interrupt ------
 1                            1000 sub  $1,%dx
 1   ------ Interrupt ------  ------ Interrupt ------
 1   1001 test $0,%dx
 1   1002 jgte .top
 0   1000 sub  $1,%dx
 1   ------ Interrupt ------  ------ Interrupt ------
 1                            1001 test $0,%dx
 1                            1002 jgte .top
 0                            1000 sub  $1,%dx
 0   ------ Interrupt ------  ------ Interrupt ------
 0   1001 test $0,%dx
 0   1002 jgte .top
-1   1000 sub  $1,%dx
 0   ------ Interrupt ------  ------ Interrupt ------
 0                            1001 test $0,%dx
-1   ------ Interrupt ------  ------ Interrupt ------
-1   1001 test $0,%dx
-1   1002 jgte .top
 0   ------ Interrupt ------  ------ Interrupt ------
 0                            1002 jgte .top
-1                            1000 sub  $1,%dx
-1   ------ Interrupt ------  ------ Interrupt ------
-1   1003 halt
-1   ----- Halt;Switch -----  ----- Halt;Switch -----
-1                            1001 test $0,%dx
-1   ------ Interrupt ------  ------ Interrupt ------
-1                            1002 jgte .top
-1   ------ Interrupt ------  ------ Interrupt ------
-1                            1003 halt
```

可以看到，不管中断间隔的大小是多少，不管是否固定，我们最终得到的结果和具体执行的代码是相同的。因为这两个子进程并不共享变量，没有临界区，所以两个子进程交替运行不会产生竞争条件。

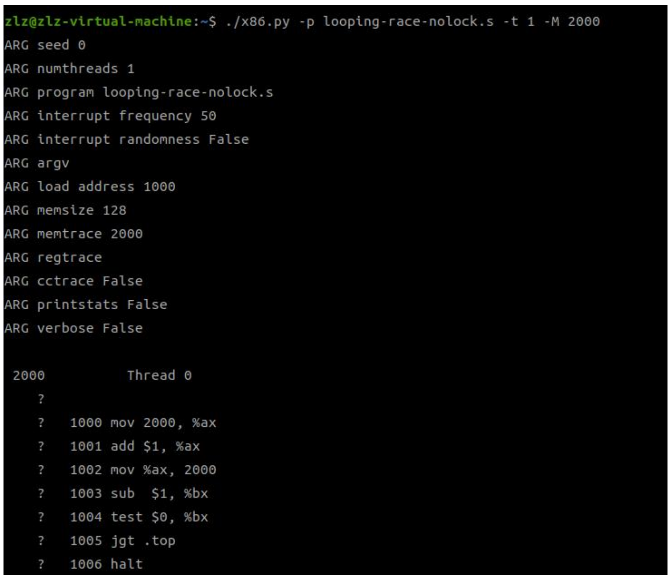
4.接下来我们将研究一个不同的程序（looping-race-nolock.s）,该程序访问位于内存地址 2000 的共享变量，简单起见，我们称这个变量为 x.使用单线程运行它，并确保你了解它的功能，如下所示:
./x86.py -p looping-race-nolock.s -t 1 -M 2000
在整个运行过程中， x（即内存地址为 2000）的值是多少？使用-c 来检查你的答案。

首先，查看 looping-race-nolock.s 的内容



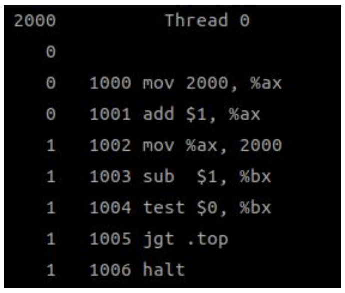
```
# assumes %bx has loop count in it

.main
.top
# critical section
mov 2000, %ax  # get 'value' at address 2000
add $1, %ax    # increment it
mov %ax, 2000  # store it back

# see if we're still looping
sub  $1, %bx
test $0, %bx
jgt .top

halt
```

将地址为 2000 处的值给寄存器 ax，然后将 ax 的值+1，然后将 ax 的值给到地址为 2000 的地方，将寄存器 bx 的值-1，然后 bx 与 0 取与，如果大于那么就跳转到.top



```
zlz@zlz-virtual-machine:~$ ./x86.py -p looping-race-nolock.s -t 1 -M 2000
ARG seed 0
ARG numthreads 1
ARG program looping-race-nolock.s
ARG interrupt frequency 50
ARG interrupt randomness False
ARG argv
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace
ARG cctrace False
ARG printstats False
ARG verbose False

 2000          Thread 0
    ?
    ?   1000 mov 2000, %ax
    ?   1001 add $1, %ax
    ?   1002 mov %ax, 2000
    ?   1003 sub  $1, %bx
    ?   1004 test $0, %bx
    ?   1005 jgt .top
    ?   1006 halt
```

通过-c 查看结果



```
 2000          Thread 0
    0
    0   1000 mov 2000, %ax
    0   1001 add $1, %ax
    1   1002 mov %ax, 2000
    1   1003 sub  $1, %bx
    1   1004 test $0, %bx
    1   1005 jgt .top
    1   1006 halt
```

这段代码是有竞争条件的，因为如果两个相同的线程运行，就会进行 mov %ax,2000，而这个 2000 的地址处是我们线程的共享数据资源，如果两个线程同时进入临界区就会导致错误。可能发生的错误如下：

线程 1	线程 2	地址 2000 处的数据	ax 寄存器
mov 2000, %ax add $1, %ax		x	x+1
	mov 2000, %ax add $1, %ax mov %ax, 2000	x+1	x+1
mov %ax, 2000		x+1	x+1

在两个线程各自的一次循环中，发生了调度，本应在两个分别的循环实现 x=x+2，产生了错误的结果 x=x+1，这就是此处竞态条件可能导致的错误。