Computer Architecture
DAT105
Exercise Session 4

Piyumal Ranawka
piyumal@chalmers.se
M. Waqar Azhar
waqarm@chalmers.se

Chalmers University of Technology, Sweden

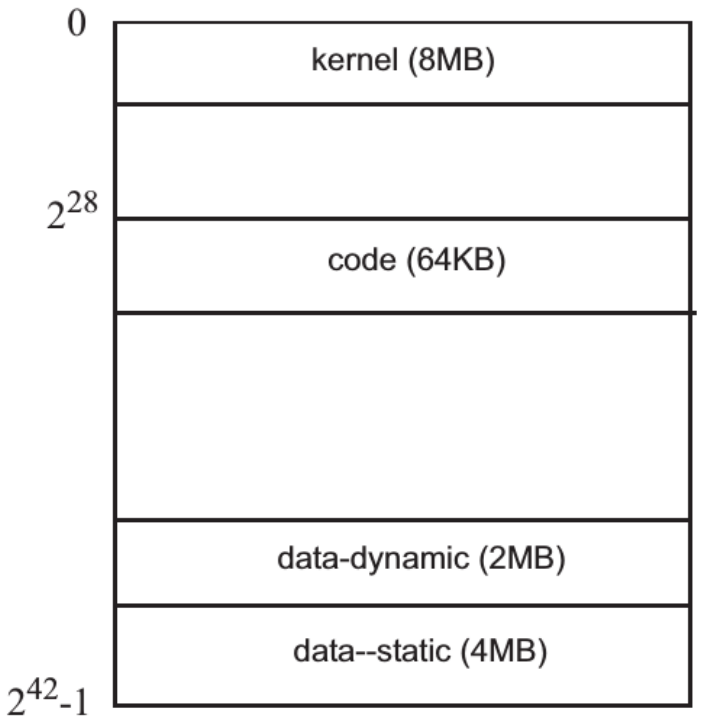
October 1, 2021

Table of Contents

Table of Contents

Problem 4.3

This problem is about the structure of page tables to support large virtual address spaces. Assume a 42-bit virtual address space per process in a 64-bit machine and 512 MByte of main memory. The page size is 4KBytes. Page table entries are 4 byte in every table. Various hierarchical page table organizations are envisioned: 1, 2, and 3 levels. The virtual space to map is populated as shown in Figure 4.21. Kernel space addresses are not translated because physical addresses are identical to virtual addresses. However virtual addresses in all other segments must be translated.



Problem 4.3 (Part-1)

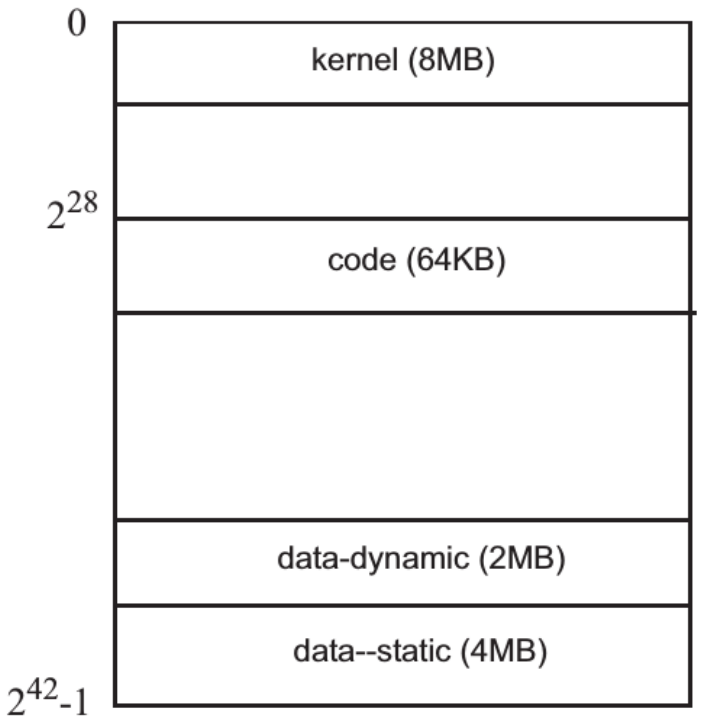
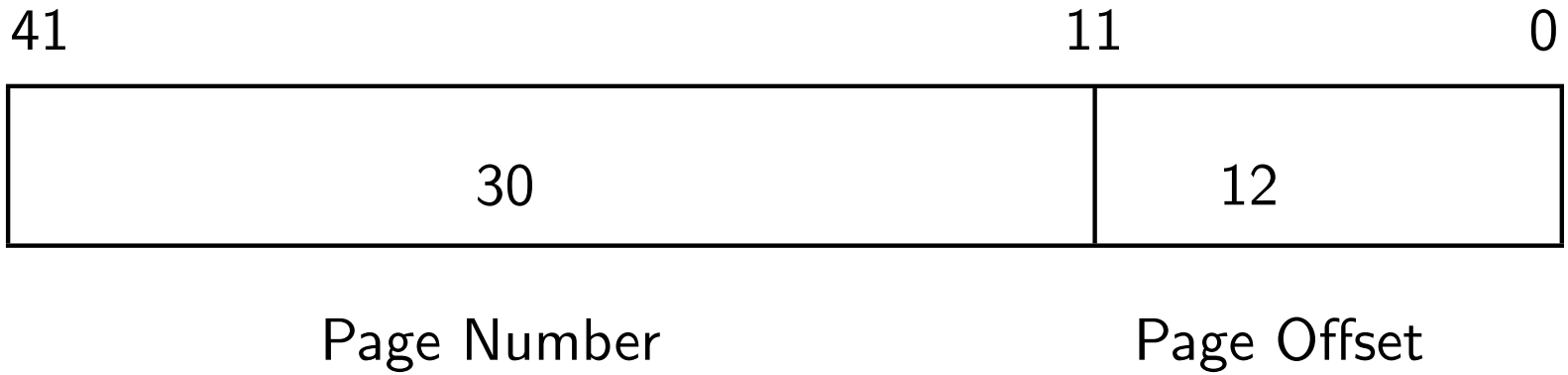


Figure 4.21. Virtual address space mapping

Q1 : What would be the size of a single-level page table?

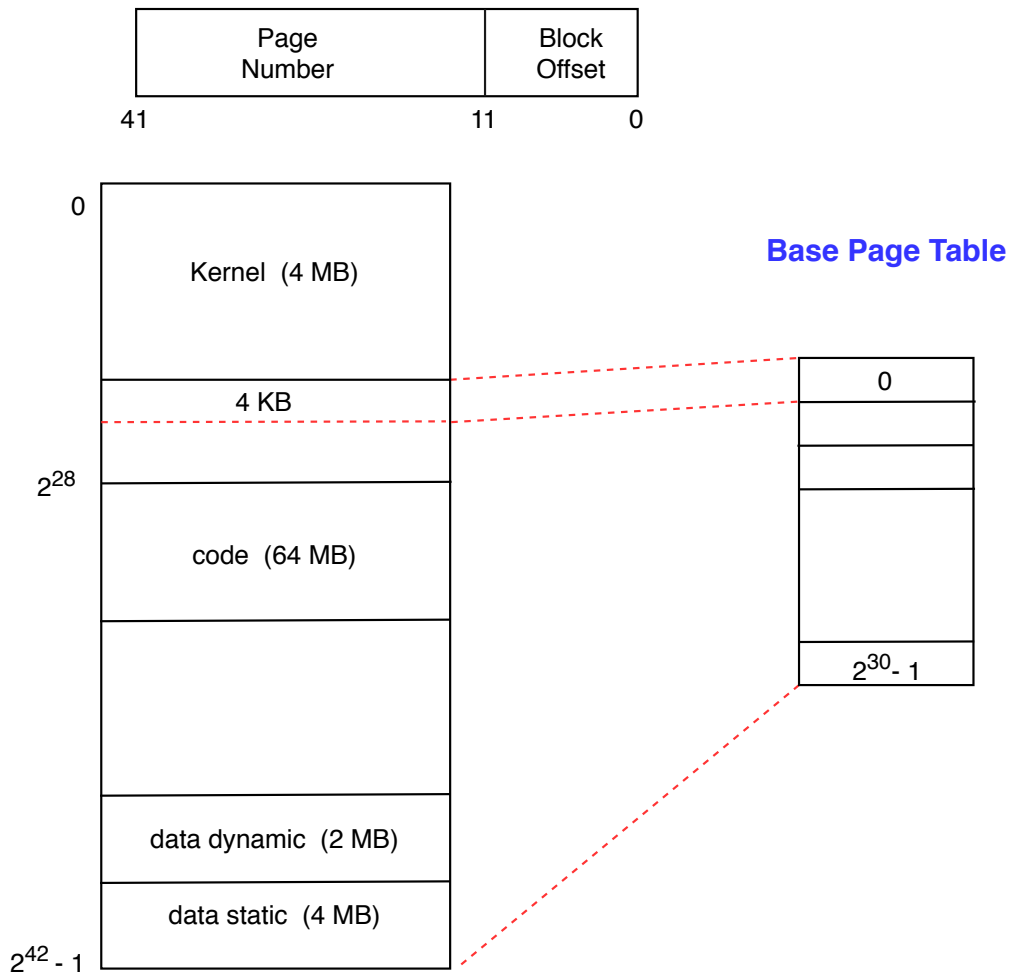
Problem 4.3 (Part-1)



Virtual address space has 42 bits.
Page size = 4 KBytes = 2^{12}
Size of each page table entry = 4 Bytes
No of pages = $2^{42}/2^{12} = 2^{42-12} = 2^{30}$
Page table size
= No of Entries * Size of Page Table Entry = $4 * 2^{30} = 4GB$

Single level page table structure

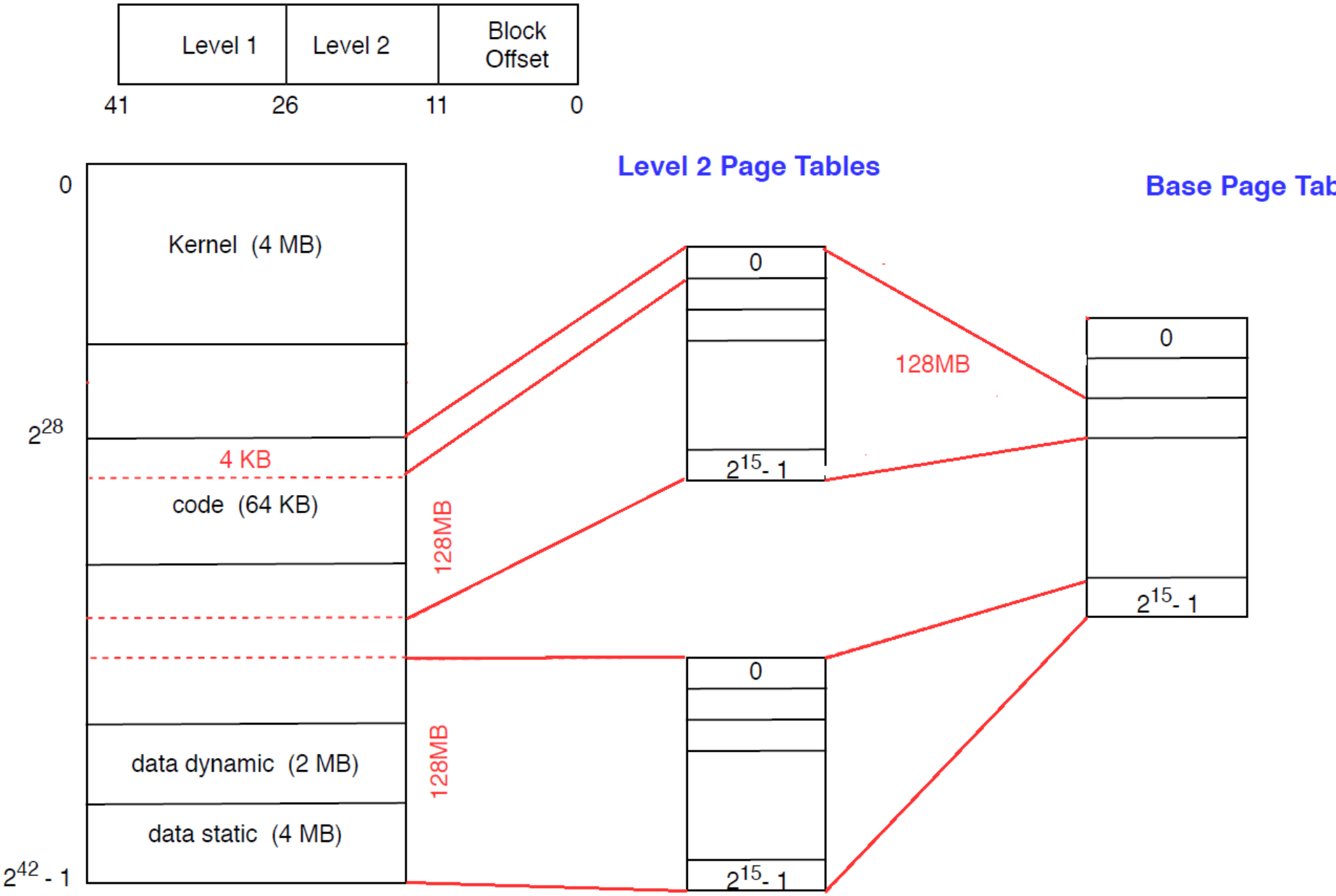
Virtual address space has 42 bits.
Page size $= 4$ KBytes $= 2^{12}$
Size of each page table entry $= 4$ Bytes
No of pages $== 2^{42}/2^{12} = 2^{42-12} = 2^{30}$
Page table size $= 4 * 2^{30} = 4GB$



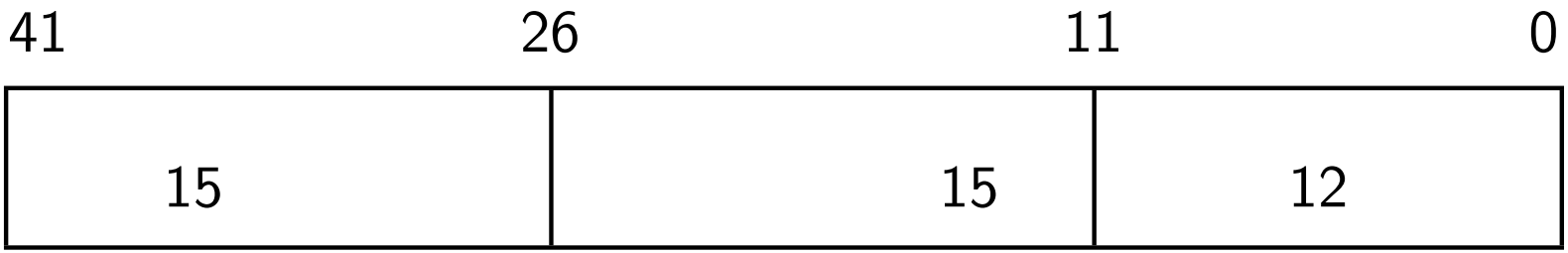
Problem 4.3 (Part-2)

Q2: Assume now a 2-level page table. We split the 30 bits of virtual page number into two fields of 15-bits each? How many page tables would we have? What would be their total size?

2-Level page table structure



Problem 4.3 (Part-2)



First Level Page Table | Second Level Page Table | Page Offset

No of entries in first level table$= 2^{15} = 2^5 * 2^{10} = 32K$
Size of each page table entry $= 4$ Bytes
Total size $= 32K * 4 = 128KB$
Each entry of the first-level page table covers
$2^{15+12} = 2^{27} = 128MB$ of virtual memory.

Problem 4.3 (Part-2)

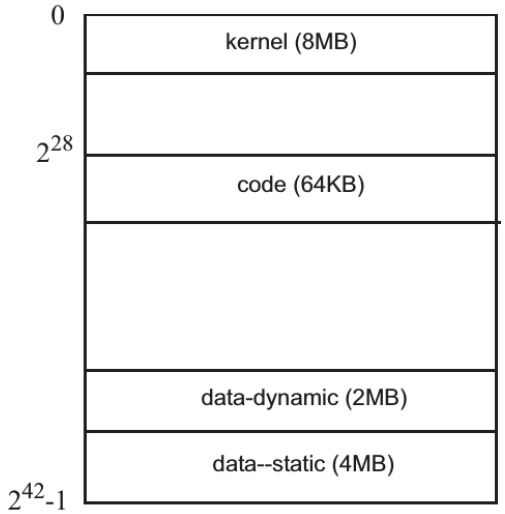
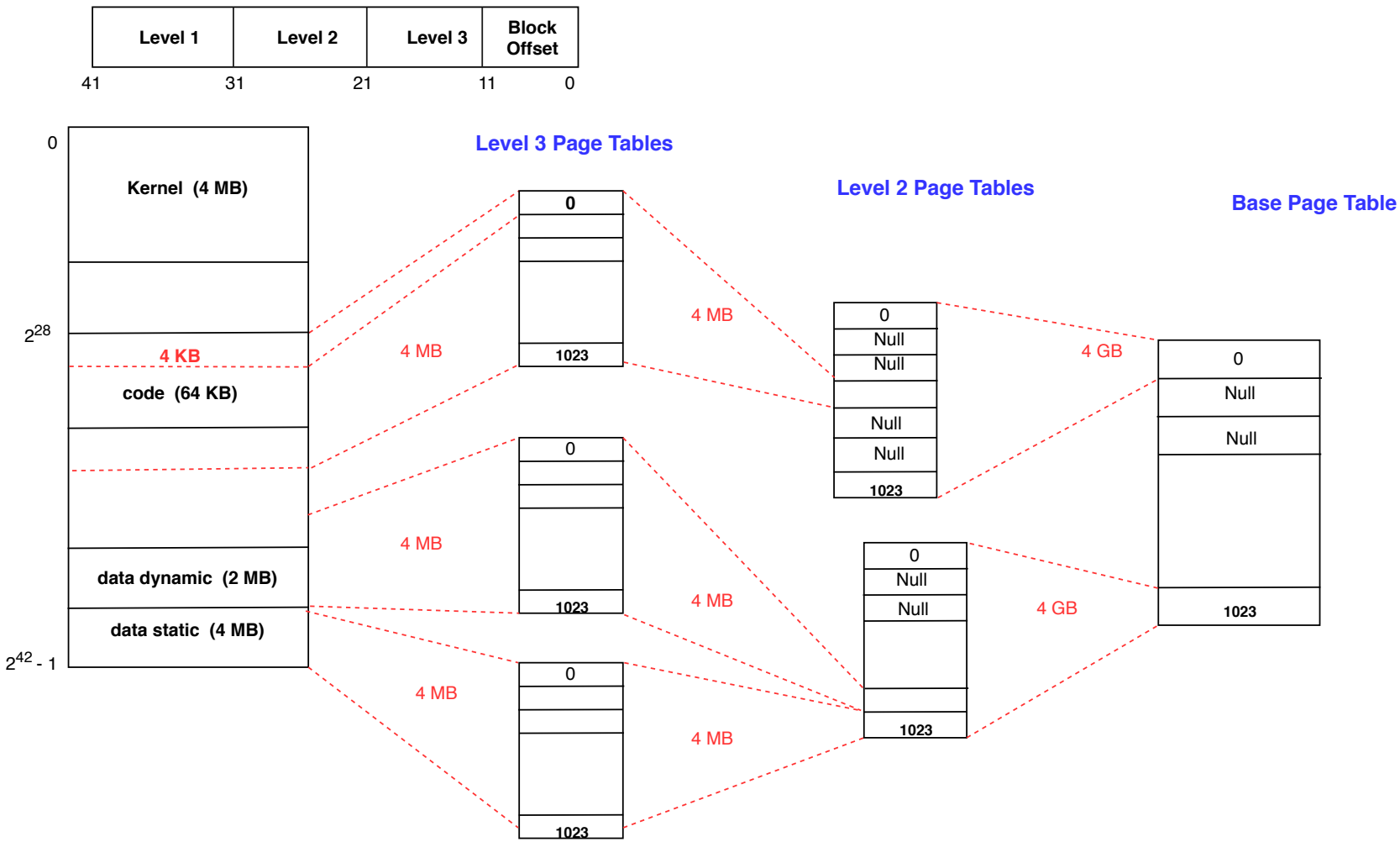


Figure 4.21. Virtual address space mapping

The code segment lies in the virtual memory area covered by the third entry of the first-level page table.
The two data segments lie in the virtual memory area covered by the last entry of the first-level page table.
Therefore we need a total of two second-level page tables which occupy 128KB each.
Thus the number of page tables is 3 and the total physical memory occupied by the page tables is $3x128KB = 384KB$.

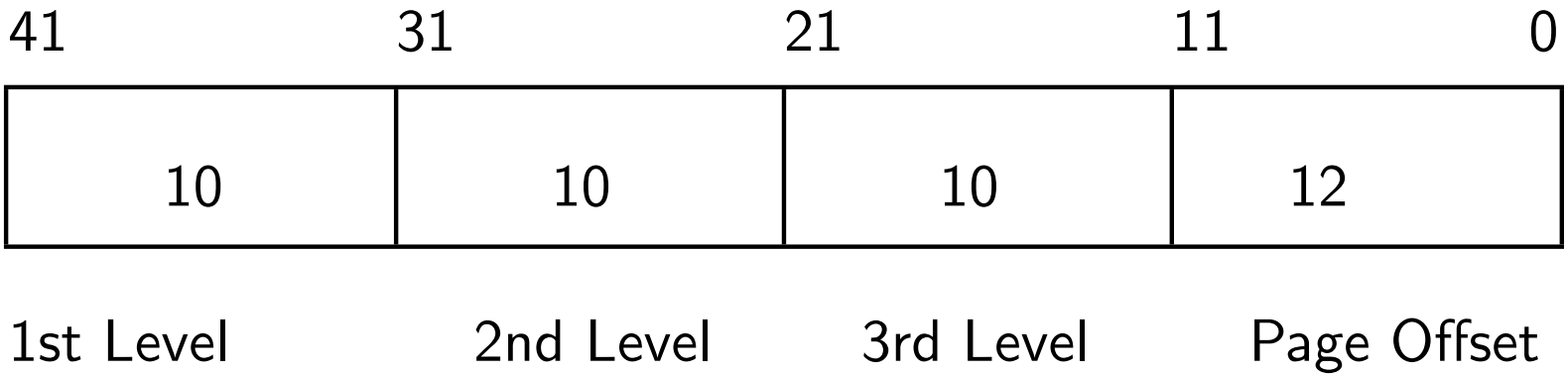
3-Level page table structure

Q3: Repeat 2. for a 3-level page table splitting the 30 bits of virtual page number into 3 fields of 10 bits each.



Problem 4.3 (Part-3)

Q3: Repeat 2. for a 3-level page table splitting the 30 bits of virtual page number into 3 fields of 10 bits each.



Level-1 table has $= 2^{10} = 1K$ entries.
Each Entry $= 4$ bytes.
Total size $= 4 * 1K = 4KB$.
A single entry of the first-level page table covers
$= 2^{10+10+12} = 2^{32} = 4GB$ of virtual space.

Problem 4.3 (Part-3)

Level-2 Page Table.
1 Table for code segment.
The two data segments lie in the virtual memory area covered by the last entry of the root page table. So we need another second level page table for the data (size 4KB).
Each entry of the second-level page tables covers
$2^{10+12} = 2^{22} = 4MB$ of virtual memory.
This is enough to cover the code segment (1-third-level table or 4KB). However the data segments requires two third-level page tables as they occupy 6MB of virtual memory.
The size of these two tables is 8KB. Thus the grand total is
$1 + 2 + 3 = 6$ tables.
Total physical memory occupation $= 6 * 4KB = 24KB$.

Table of Contents

Problem 4.5

In this problem, we explore cache mapping and cache replacement policies when memory references are cyclic or periodic. Such type of reference streams are common in accesses to instructions (loops) or in strided accesses to data.

First-level instruction caches are often direct-mapped, not only because direct-mapped caches are faster on a hit but also because they are better at handling loops than set-associative caches. Let's assume a cache with four lines (0,1,2 and 3) and a cyclic (periodic) block reference string with block addresses (0,1,2,3,4,5)10. This notation means that the reference string has a periodic pattern of accesses to block addresses 0,1,2,3,4, and 5 repeated 10 times. We classify misses into cold, capacity, and conflict misses. Capacity misses are counted in a FA cache with LRU replacement policy. In all cases the caches are empty at the beginning of the string.

Problem 4.5

Count the total number of misses in the following caches

- Direct-mapped
- FA with LRU replacement
- FA with FIFO replacement
- FA with LIFO (Last In First Out) replacement
- 2-way SA cache with LRU in each set

Problem 4.5(a) Direct-mapped Cache

Iteration	Reference	Line-0	Line-1	Line-2	Line-3	Comment
Cold	-	-	-	-	-	Cache is empty
1	0	0	-	-	-	Cold Miss
	1	0	1	-	-	Cold Miss
	2	0	1	2	-	Cold Miss
	3	0	1	2	3	Cold Miss
	4	4	1	2	3	Cold Miss
	5	4	5	2	3	Cold Miss
2	0	0	5	2	3	Miss
	1	0	1	2	3	Miss
	2	0	1	2	3	Hit
	3	0	1	2	3	Hit
	4	4	1	2	3	Miss
	5	4	5	2	3	Miss
3	0	0	5	2	3	Miss
	1	0	1	2	3	Miss
	2	0	1	2	3	Hit
	3	0	1	2	3	Hit
	4	4	1	2	3	Miss
	5	4	5	2	3	Miss
4	0	0	5	2	3	Miss
	1	0	1	2	3	Miss
	2	0	1	2	3	Hit
	3	0	1	2	3	Hit
	4	4	1	2	3	Miss
	5	4	5	2	3	Miss

$$\text{Total Misses} = 6 + 9 * 4 = 42$$

Problem 4.5(a) F-A Cache with LRU

Iteration	Reference	Line-0	Line-1	Line-2	Line-3	Comment
Cold	-	-	-	-	-	Cache is empty
1	0	0	-	-	-	Cold Miss
	1	0	1	-	-	Cold Miss
	2	0	1	2	-	Cold Miss
	3	0	1	2	3	Cold Miss
	4	4	1	2	3	Cold Miss
	5	4	5	2	3	Cold Miss
2	0	4	5	0	3	Miss
	1	4	5	0	1	Miss
	2	2	5	0	1	Miss
	3	2	3	0	1	Miss
	4	2	3	4	1	Miss
	5	2	3	4	5	Miss
3	0	0	3	4	5	Miss
	1	0	1	4	5	Miss
	2	0	1	2	5	Miss
	3	0	1	2	3	Miss
	4	4	1	2	3	Miss
	5	4	5	2	3	Miss
4	0	4	5	0	3	Miss
	1	4	5	0	1	Miss
	2	2	5	0	1	Miss
	3	2	3	0	1	Miss
	4	2	3	4	1	Miss
	5	2	3	4	5	Miss

$$\text{Total Misses} = 6 + 9 * 6 = 60$$

Problem 4.5(a) F-A Cache with FIFO

Iteration	Reference	Line-0	Line-1	Line-2	Line-3	Comment
Cold	-	-	-	-	-	Cache is empty
1	0	0	-	-	-	Cold Miss
	1	0	1	-	-	Cold Miss
	2	0	1	2	-	Cold Miss
	3	0	1	2	3	Cold Miss
	4	4	1	2	3	Cold Miss
	5	4	5	2	3	Cold Miss
2	0	4	5	0	3	Miss
	1	4	5	0	1	Miss
	2	2	5	0	1	Miss
	3	2	3	0	1	Miss
	4	2	3	4	1	Miss
	5	2	3	4	5	Miss
3	0	0	3	4	5	Miss
	1	0	1	4	5	Miss
	2	0	1	2	5	Miss
	3	0	1	2	3	Miss
	4	4	1	2	3	Miss
	5	4	5	2	3	Miss
4	0	4	5	0	3	Miss
	1	4	5	0	1	Miss
	2	2	5	0	1	Miss
	3	2	3	0	1	Miss
	4	2	3	4	1	Miss
	5	2	3	4	5	Miss

$$\text{Total Misses} = 6 + 9 * 6 = 60$$

Problem 4.5(a) F-A Cache with LIFO

Iteration	Reference	Line-0	Line-1	Line-2	Line-3	Comment
Cold	-	-	-	-	-	Cache is empty
1	0	0	-	-	-	Cold Miss
	1	0	1	-	-	Cold Miss
	2	0	1	2	-	Cold Miss
	3	0	1	2	3	Cold Miss
	4	0	1	2	4	Cold Miss
	5	0	1	2	5	Cold Miss
2	0	0	1	2	5	Hit
	1	0	1	2	5	Hit
	2	0	1	2	5	Hit
	3	0	1	2	3	Miss
	4	0	1	2	4	Miss
	5	0	1	2	5	Miss
3	0	0	1	2	5	Hit
	1	0	1	2	5	Hit
	2	0	1	2	5	Hit
	3	0	1	2	3	Miss
	4	0	1	2	4	Miss
	5	0	1	2	5	Miss
4	0	0	1	2	5	Hit
	1	0	1	2	5	Hit
	2	0	1	2	5	Hit
	3	0	1	2	3	Miss
	4	0	1	2	4	Miss
	5	0	1	2	5	Miss

$$\text{Total Misses} = 6 + 9 * 3 = 33$$

Problem 4.5(a) 2 way S-A Cache with LRU per set

Iteration	Reference	Set-1		Set-2		Comment
		Line-0	Line-1	Line-2	Line-3	
Cold	-	-	-	-	-	Cache is empty
1	0	0	-	-	-	Cold Miss
	1	0	-	1	-	Cold Miss
	2	0	2	1	-	Cold Miss
	3	0	2	1	3	Cold Miss
	4	4	2	1	3	Cold Miss
	5	4	2	5	3	Cold Miss
2	0	4	0	5	3	Miss
	1	4	0	5	1	Miss
	2	2	0	5	1	Miss
	3	2	0	3	1	Miss
	4	2	4	3	1	Miss
	5	2	4	3	5	Miss
3	0	0	4	3	5	Miss
	1	0	4	1	5	Miss
	2	0	2	1	5	Miss
	3	0	2	1	3	Miss
	4	4	2	1	3	Miss
	5	4	2	5	3	Miss
4	0	4	0	5	3	Miss
	1	4	0	5	1	Miss
	2	2	0	5	1	Miss
	3	2	0	3	1	Miss
	4	2	4	3	1	Miss
	5	2	4	3	5	Miss

Total Misses $= 6 + 9 * 6 = 60$

Problem 4.5(b)COLD MISSES

The number of cold misses is independent of the cache organization and replacement policy. The number of cold misses is equal to the number of blocks in the trace

Number pf cold misses = 6 misses

Problem 4.5(b)CAPACITY MISSES using FA-LRU

The number of capacity misses is also independent of the cache organization and replacement policy.

Capacity misses = Number of misses in an FA LRU cache - Number of cold misses

Number capacity misses = 60 - 6 = 54 misses

Problem 4.5(b)CONFLICT MISSES using FA-LRU

conflict misses = total number of misses - cold misses - capacity misses

Direct-mapped: Number of conflict misses = 42 - (6 + 54) = -18 misses

FA with LRU: Number of conflict misses = 60 - (6 + 54) = 0

FA with FIFO: Number of conflict misses = 60 - (6 + 54) = 0

FA with LIFO: Number of conflict misses = 33 - (6 + 54) = -27 misses

2-way SA with LRU for each set: Number of conflict misses = 60 - (6 + 54) = 0

Problem 4.5(a) FA with OPT

Iteration	Reference	Line-0	Line-1	Line-2	Line-3	Comment
Cold	-	-	-	-	-	Cache is empty
1	0	0	-	-	-	Cold Miss
	1	0	1	-	-	Cold Miss
	2	0	1	2	-	Cold Miss
	3	0	1	2	3	Cold Miss
	4	0	1	2	4	Cold Miss
	5	0	1	2	5	Cold Miss
2	0	0	1	2	5	Hit
	1	0	1	2	5	Hit
	2	0	1	2	5	Hit
	3	0	1	3	5	Miss
	4	0	1	4	5	Miss
	5	0	1	4	5	Hit
3	0	0	1	4	5	Hit
	1	0	1	4	5	Hit
	2	0	2	4	5	Miss
	3	0	3	4	5	Miss
	4	0	3	4	5	Hit
	5	0	3	4	5	Hit
4	0	0	3	4	5	Hit
	1	1	3	4	5	Miss
	2	2	3	4	5	Miss
	3	2	3	4	5	Hit
	4	2	3	4	5	Hit
	5	2	3	4	5	Hit

Problem 4.5(a) FA with OPT

First 6 accesses are cold misses (1-6 memory access) .

From remaining $60 - 6 = 54$ access, we have pattern where there are 3-hits and 2-misses every 5 accesses.

So, $54/5 = 10$ pattern plus 4 spare accesses.

So for the 10 access patterns $10 * 2 = 20$ misses (7-54 memory access)

The hit-miss pattern in each of these 5-access pattern is 3-hits and 2-misses, so in last 4-accesses(55-60 memory access) , there will be 3-hits and 1-miss.

So total misses $= 6 + 20 + 1 = 27$

Problem 4.5(b)CONFLICT MISSES using FA-Opt

Total Misses FA-Opt= 27

Direct-mapped: Number of conflict misses = 42 - 27 = 15 misses

FA with LRU: Number of conflict misses = 60 - 27 = 33 misses

FA with FIFO: Number of conflict misses = 60 - 27 = 33 misses

FA with LIFO: Number of conflict misses = 33 - 27 = 6 misses

2-way SA with LRU for each set: Number of conflict misses = 60 - 27 = 33 misses

Table of Contents

Problem 4.8 (Problem Statement)

A simple program that accumulates values from a vector in memory is used in this problem

```
LOOP :
LW R4,0(R3)
ADDI R3,R3,stridex4    // stridex4: stride multiplied by 4
ADD R1,R1,R4
BNE R3,R5,LOOP
```

The stride is the difference between the indexes of two consecutive vector elements. It is multiplied by 4 to find the address of consecutive vector components.

Problem 4.8

We examine the efficiency of this loop on the 5-stage pipeline with a blocking and a nonblocking data cache. The branch flushes the I-fetch and I-decode stages whenever it is taken.

Throughout this problem the data cache is direct-mapped and is empty at the start and its block size is 64bytes. Assume that the number of iterations of this loop is extremely large (the number of components to add is in the millions).

At first let the latency of an L1 miss be 30 clocks. Remember that the LW is first tried in the cache. If it misses, the pipeline "freezes" for 30 clocks and then the pipeline is restarted.

Problem 4.8 (Part-a)

Consider first that the cache is blocking. Find the average execution time (in cycles) of each iteration of the loop (i.e., the average time taken by each accumulation) as a function of the stride.

Problem 4.8 (Execution times for Hit & Miss cases)

In case of a hit

LOOP:	LW R4,0(R3)	1 cycle
	ADDI R3,R3,stridex4	1 cycle
	ADD R1,R1,R4	1 cycle
	BNE R3,R5,LOOP	1 cycle + 2 cycle for pipeline flush

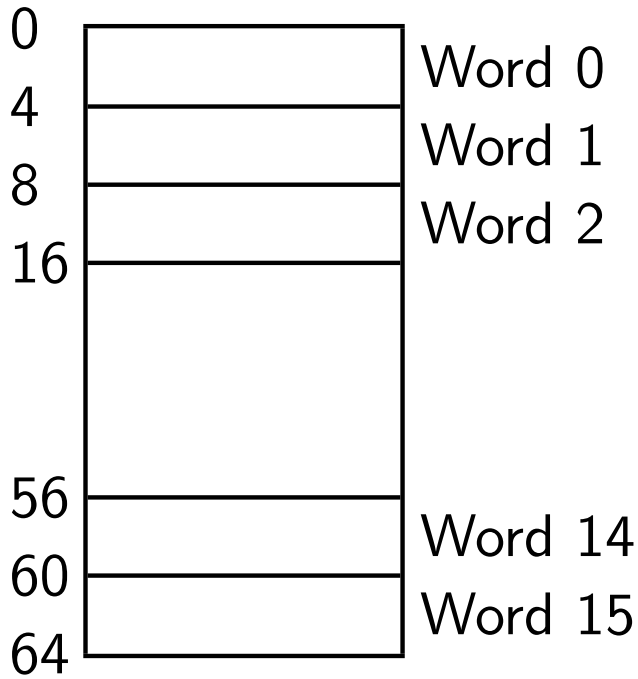
Execution Time (hit in cache) = 6

In case of a miss

LOOP:	LW R4,0(R3)	1 cycle
	ADDI R3,R3,stridex4	1 cycle
	ADD R1,R1,R4	1 cycle + 30 cache latency + 1 retry
	BNE R3,R5,LOOP	1 cycle + 2 cycle for pipeline flush

Execution Time (Miss in cache) = 37

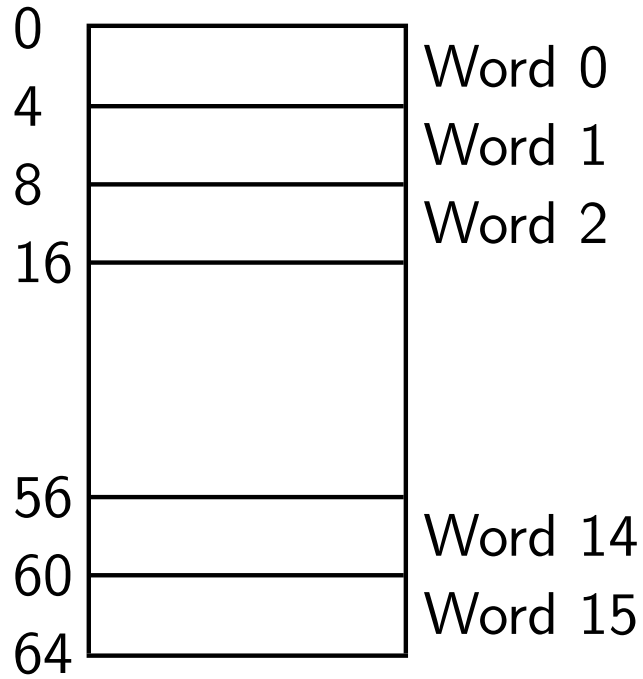
Problem 4.8(Hit Rate)



64-Byte Cache Block

Problem 4.8

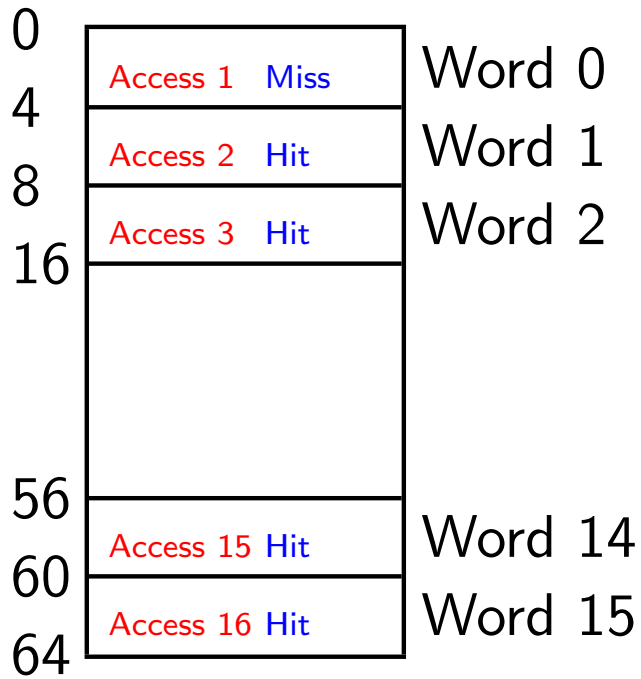
```
LOOP :
LW R4,0(R3)
ADDI R3,R3,stridex4
ADD R1,R1,R4
BNE R3,R5,LOOP
```



Problem 4.8

Stride = 1

```
LOOP :
LW R4,0(R3)
ADDI R3,R3,4
ADD R1,R1,R4
BNE R3,R5,LOOP
```

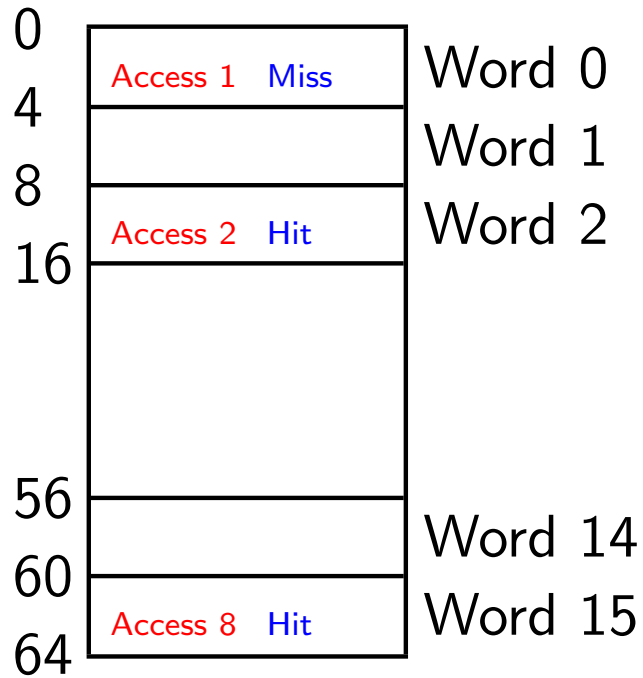


1 miss and 15 Hits

Problem 4.8

Stride = 2

```
LOOP :
LW R4,0(R3)
ADDI R3,R3,8
ADD R1,R1,R4
BNE R3,R5,LOOP
```

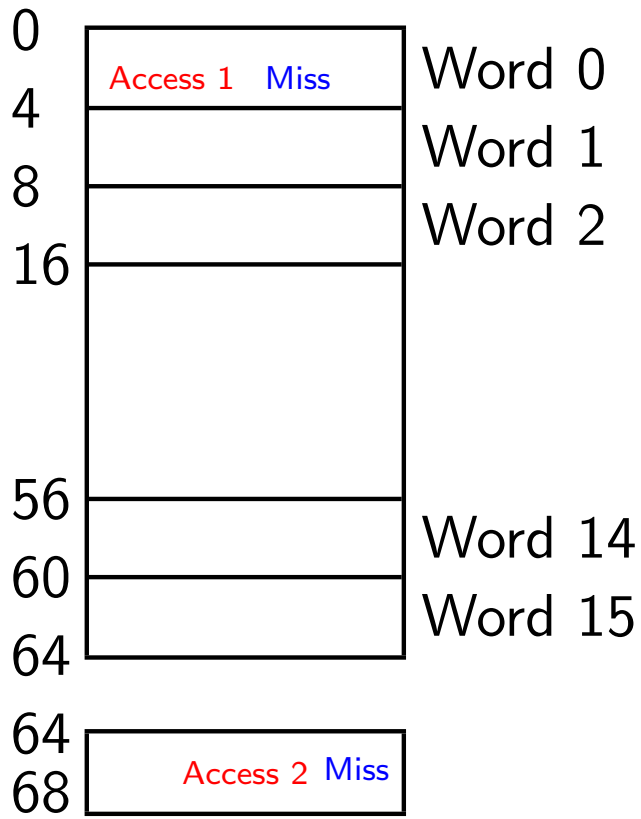


1 miss and 7 Hits

Problem 4.8

Stride = 16

```
LOOP :
LW R4,0(R3)
ADDI R3,R3,64
ADD R1,R1,R4
BNE R3,R5,LOOP
```



All Accesses will miss

Problem 4.8

Stride >= 16

We always miss

Average Execution time = 37 cycles

Problem 4.8

Stride < 16

Number of accesses to single block $= \frac{LCM(16,stride)}{stride}$

Number of misses $= \frac{LCM(16,stride)}{16}$

LCM(a, b), is the smallest positive integer that is divisible by both a and b

Problem 4.8

$$\text{Avergae execution Time} = \frac{Misses*37+Hits*6}{Iterations} = \frac{Misses*37+Hits*6}{Accesses}$$

$$\text{Avergae execution Time} = \frac{Misses*37+(Accesses-Misses)*6}{Accesses}$$

$$\text{Avergae execution Time} = \frac{\frac{LCM(16,stride)}{16}*37+(\frac{LCM(16,stride)}{stride}-\frac{LCM(16,stride)}{16})*6}{\frac{LCM(16,stride)}{stride}}$$

$$\text{Avergae execution Time} = \frac{37*stride}{16}+6-\frac{6*stride}{16} = \frac{31}{16}*stride+6$$

Problem 4.8 (Part b)

The next idea we explore is that of a very simple lock up free (non blocking) cache, which can execute a load hit while a (non-blocking) prefetch miss is pending in the cache. A prefetch executes like a Load in the pipeline except that 1) it does not return a value and 2) it never blocks the pipeline even on a cache prefetch miss. Now, whenever a Load misses, the pipeline freezes until the miss is resolved or a pending missing prefetch is returned in the cache. At that time the load is retried. The new prefetch instruction is PW d(R), where d is the displacement and R is a base address. Whenever a prefetch is a primary miss, the cache controller fetches the block from the next cache level. Whenever a prefetch hits in cache or is a secondary miss, the cache controller drops the prefetch. All prefetches meeting an exception are also dropped. The following code is proposed

```
LOOP :
LW R4,0(R3)
PW stridex4(R3)
ADDI R3,R3,stridex4
ADD R1,R1,R4
BNE R3,R5,LOOP
```

Problem 4.8 (Execution times for Hit & Miss)

In Case of a Hit !

LOOP:	LW R4,0(R3)	1-cycle
	PW stridex4(R3)	1-cycle
	ADDI R3,R3,stridex4	1-cycle
	ADD R1,R1,R4	1-cycle
	BNE R3,R5,LOOP	1-cycle + 2 cycle for pipeline flush

Execution Time for Hit Iteration = 7

In Case of a miss !

LOOP:	LW R4,0(R3)	1-cycle
	PW stridex4(R3)	1-cycle
	ADDI R3,R3,stridex4	1-cycle
	ADD R1,R1,R4	1-cycle + 30 + 1- 6
	BNE R3,R5,LOOP	1-cycle + 2-cycle for pipeline flush

Execution Time for Miss Iteration = 37+1-6= 32

Problem 4.8 (Part-b)

Stride >= 16

We always miss

Average Execution time = 32 cycles

Problem 4.8(Part-b)

Stride < 16

$$\text{Avergae execution Time} = \frac{Misses*32+Hits*7}{Iterations} = \frac{Misses*32+Hits*7}{Accesses}$$

$$\text{Avergae execution Time} = \frac{Misses*32+(Accesses-Misses)*7}{Accesses}$$

$$\text{Avergae execution Time} = \frac{\frac{LCM(16,stride)}{16}*32+(\frac{LCM(16,stride)}{stride}-\frac{LCM(16,stride)}{16})*7}{\frac{LCM(16,stride)}{stride}}$$

$$\text{Avergae execution Time} = \frac{25}{16} * stride + 7$$

Problem 4.8 (Part-c)

Is it possible to find out when the prefetch is effective, as a function of the stride, the block size and the miss latency? What is the breakeven point in general?

Problem 4.8 (Part-c)

Stride $>= 16$

We always miss

prefetch is always more effective

Problem 4.8 (Part-c)

Stride < 16

Avergae execution Time with prefecth $<$ Avergae execution Time with out prefech

$\frac{25}{16} * stride + 7 < \frac{31}{16} * stride + 6$

$stride >= 3$