c. First-Level Cache
The first level cache is a virtually indexed, physically tagged 3-way cache.
First-level Cache size: 96KB
Block size: 16bytes
Set size: 3 blocks per set
The width of each data ram column is 64 bits.
1. Which bits of the virtual address are used to index the tag RAM of the directory?
2. Which bits of the virtual address are used to index the data RAM of the directory?
3. Which bits of the physical address are matched against the tags in the Tag RAMs?
4. In this first-level cache, there are consistency problems due to synonyms. One solution is to search all the sets that could contain the block on every miss. How many sets should be searched?
5. Another solution to solve the synonym problem is page coloring. What are the bits defining the color of the page?

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.

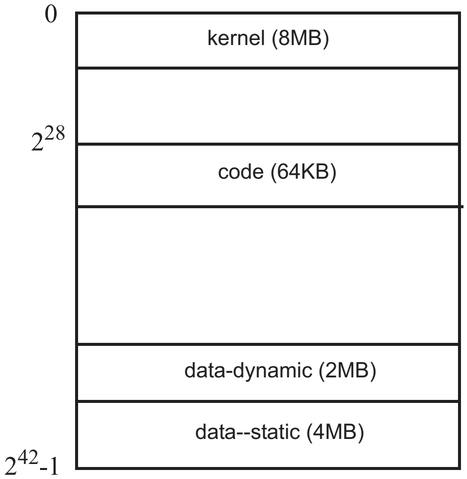

Figure 4.21. Virtual address space mapping

Please answer the following questions:
1. What would be the size of a single-level page table?
2. 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?
3. Repeat 2. for a 3-level page table splitting the 30 bits of virtual page number into 3 fields of 10 bits each.

Problem 4.4

What is pseudo-LRU? Search online for a paper describing pseudo LRU, describe how it works for a cache of 4 lines and explain its advantages over pure LRU.

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.

a. 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, and a 2-way SA cache with LRU in each set.
b. Based on your results in a., what is the number of cold, capacity and conflict misses for each cache.
c. Consider now the optimum replacement policy in the FA cache. An optimum policy is guaranteed to give the maximum hit rate, although it is not feasible in practice. The optimum policy replaces the block in cache which will be referenced the farthest away in the future. We define the forward distance of a block i as the number of *distinct* block numbers between the current reference and the next reference to the block. Thus we replace the block that has largest forward distance. Imagine that all blocks are referenced one extra time at the end of the trace. If the forward distances of several blocks are equal, pick anyone of them for replacement. Thus there may be different sequences of replaced blocks but they will all have the same miss rate. What is the number of misses in the FA cache with OPT replacement?
d. For all the caches in a., compute the number of conflict misses when the FA cache used in the count of capacity misses is the FA cache with OPT replacement.

Problem 4.6

In this problem we compare cache replacement policies for a given trace of accesses to a set. A trace is made of consecutive block addresses dynamically accessed by a program.
Take the following trace, where each letter is a block address:

aabcaadeffefefefabgcaef

a. Assume that the fully associative cache of size 4 lines is cold at the beginning. What is the miss rate under the following replacement policies:

- LRU
- FIFO (First in first out)
- Pseudo-LRU (find out what that is online)
- LFU (Least Recently used). In this policy, the replacement policy counts the number of accesses to each block. The block with the least number of references is replaced. In case of a tie, any one of the LFU blocks is picked at random. To make the solution uniform, pick the LRU block among the LFU blocks.
- Optimum

b. What is the number of conflict misses for each cache in the following cases:

- The number of capacity misses is counted for a FA cache with LRU

- The number of capacity misses is counted for a FA cache with OPT

Problem 4.7

Most scientific applications involve matrix operations. The most common operations is matrix multiply:

```
X = Y*Z
```

where X, Y, and Z are N-by-N matrices.

If the elements of X are computed row-by-row and the L1 cache is too small to hold Z, the entire matrix Z must be reloaded in cache (capacity misses) for each row of X, resulting in a total number of cache misses of the order of N^3. To reduce the miss rate in the cache the compiler can block the matrix multiply.
Assume a fully associative data cache with a LRU (least recently used) replacement policy. Matrix sizes are 256x256. Each element is a double precision floating point number (8 bytes). To simplify, assume that the cache block size is 8 bytes as well and that operands are aligned.
1. Compute the total number of misses in the data cache, assuming that the data cache size is 128KB. To compute the data cache miss rate, we neglect all integer accesses in the program execution and focus on the floating point accesses. Also compute the total number of operations (FP ADDs and FP MULT).

2. To reduce the miss rate, a PhD student decides that restructuring the algorithm would be a good idea. The student believes that by blocking the matrix multiply into operations on submatrices of size N/k by N/k (k is a power of 2), the miss rate and the execution time will be much improved because several submatrices easily fit in the cache now.

For instance, when k=2, the blocked matrix multiply becomes a set of multiple multiplications and additions of N/2xN/2 matrices:

$$\begin{bmatrix} X11 & X12 \\ X21 & X22 \end{bmatrix} = \begin{bmatrix} Y11 & Y12 \\ Y21 & Y22 \end{bmatrix} \times \begin{bmatrix} Z11 & Z12 \\ Z21 & Z22 \end{bmatrix}$$

$$\begin{bmatrix} X11 & X12 \\ X21 & X22 \end{bmatrix} = \begin{bmatrix} Y11xZ11 + Y12xZ21 & Y11xZ12 + Y12xZ22 \\ Y21xZ11 + Y22xZ21 & Y21xZ12 + Y22xZ22 \end{bmatrix}$$

Assuming again that the cache size is 128KB, describe the blocking algorithm that should be employed to minimize the number of misses, and compute the overall number of misses. Also compute the total number of operations (FP ADDs and FP MULT).

3. Considering both the amount of chip I/O due to cache misses and the total amount of compute operations, discuss whether it is a good idea to listen to that PhD student?

Problem 4.8

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.

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.

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.

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
```

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.

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?

d. Let's assume that the stride is 1 word (4 bytes). The following code prefetches when the address is a multiple of 64

Should be 00111111

a binary number is a multiple of 64 when all the first 6 Lsb bits are zero

```
    LOOP :  LW R4,0(R3)
            ANDI R10,R3,0x01000000
            BNEZ R10, prefetch
            ADDI R3,R3,#4
            ADD R1,R1,R4
            BNE R3,R5,LOOP
prefetch:   PW 64(R3)
            ADDI R3,R3,#4
            ADD R1,R1,R4
            BNE R3,R5,LOOP
```

should be BEZ

Can this code be better than the code in part b? When?