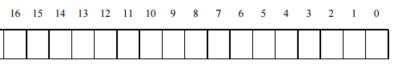
Problem 1:

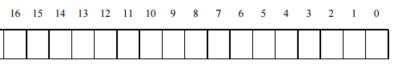


TLBI (6-8)

TLBT

VPO

VPN



CO

CI

CT

PPO

PPN

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Virtual address | 0x3925 | 0x1330 | 0x1383 | 0x0766 |
| Virtual address | 0011 1001 0010 0101 | 0001 0011 0011 0000 | 0001 0011 1000 0011 | 0000 0111 0110 0110 |
| Virtual page number | 0x1C | 0x9 | 0x9 | 0x3 |
| TLB index | 0x0 | 0x1 | 0x1 | 0x1 |
| TLB tag | 0xe | 0x4 | 0x4 | 0x1 |
| TLB hit? | Yes | No | No | yes |
| Page fault? | No | No | No | No |
| Physical page number | 0x0 | 0x8 | 0x8 | 0x5 |
| Physical address | 0 0000 0010 0101 | 1 0001 0011 0000 | 1 0001 1000 0011 | 0 1011 0110 0110 |
| Byte offset | 0x1 | 0x0 | 0x3 | 0x2 |
| Cache index | 0x1 | 0x4 | 0x0 | 0x1 |
| Cache tag | 0x1 | 0x89 | 0x8c | 0x5B |
| Cache hit? | no | no | no | no |
| Byte read from cache |  |  |  |  |

Problem 2:

To finish \*alloc we must first set ret to the first node in the free list then remove the node from the list because it will no longer be a free object. Thus we set the ret pointer to the first node in the list (the one immediately after root), then change root.next\_free to the 2nd node in the free list (excluding the root) thus removing the first node from the list.

Remainder of \*alloc:

\*ret = &root;

\*root.next\_free = &ret.next\_free;

To finish dealloc we must add the deallocated object to the free list, we’ll add it to the back of the list thus we must traverse the free list until the end where there will be a null pointer, then change that pointer so it points to our newly freed object thus adding it to the list.

Remainder of dealloc:

chunk \*temp = root;

while (temp.next\_free != null){

temp = &temp -> next\_free;

}

&temp = p;

Problem 3:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| %rax | %rbp | %rbx | %rcx | %xmm1 | %xmm0 |

|  |
| --- |
| load (vmovsd) |
| Mul (vmulsd) |
| Add (vaddsd) |
| Add (addq) |
| Cmp (cmpq) |
| Jump (jne) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| %rax | %rbp | %rbx | %rcx | %xmm1 | %xmm0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| %rax | %rbp | %rbx | %rcx | %xmm1 | %xmm0 |

b) The lower bound of the critical path for a double is (3+5)/3 = 8/3 = 2.66

c) The lower bound of the critical path for an integer is (3+1)/3 = 4/3 = 1.33

d) An iteration can overlap with the iteration before it.

For example similar to data forwarding: sum = (sum + udata[i-1] \* vdata[i-1]) + (udata [i] \* vdata[i])

Problem 4:

1. Long and double would have similar procedures for this inner product problem. Only the types would differ.

void inner4(vec\_ptr u, vec\_ptr v, data\_t \*dest)

{

long int i;

long int length = vec\_length(u);

long int length = vec\_length(v);

long int limit = length-5;

data\_t \*udata = get\_vec\_start(u);

data\_t \*vdata = get\_vec\_start(v);

data\_t sum = (data\_t) 0; // parallel accumulators

data\_t sum1 = (data\_t) 0;

data\_t sum2 = (data\_t) 0;

data\_t sum3 = (data\_t) 0;

data\_t sum4 = (data\_t) 0;

data\_t sum5 = (data\_t) 0;

//unrolls 6 at a time with 6 parallel accumulators

for (i = 0; i < limit; i = i + 6) {

sum += (udata[i] \* vdata[i]);

sum1 += (udata[i+1] \* vdata[i+1]);

sum2 += (udata[i+2] \* vdata[i+2]);

sum3 += (udata[i+3] \* vdata[i+3]);

sum4 += (udata[i+4] \* vdata[i+4]);

sum5 += (udata[i+5] \* vdata[i+5]);

}

// finishes the remainder of the elements

for (; i < limit; i++) {

sum += (udata[i] \* vdata[i]);

}

\*dest = sum + sum1 + sum2 + sum3 + sum4 + sum5;

}

1. As above, long and double are similar.

With re-association: vmovsd, vaddsd, addq, cmpq, jne

No re-association: vmovsd, vmulsd, vaddsd, addq, compq, jne

Problem 5:

Printf(“%d %d \n”, \*fmt+\*f(x), \*g(x));

2nd argument %esi

gets its value from register %eax which contains 0x20047d(%rip) + %edx

%edx is the value (%rax) which is the return value from the call to 400544 which is f(x)

0x20047d(%rip) is \*fmt

3rd argument %edx

gets its value from %ebx which gets its value from (%rax)

(%rax) is the return value from the call to 40054f which is g(x)