Problem Set 5 Solutions

Department of Electrical and Computer Engineering
The University of Texas at Austin
EE 382N.1, Fall 2018
Instructor: Dam Sunwoo
TA: Pritesh Chhajed

Questions

Problem 1

Consider the following piece of code:

```
for(i = 0; i < 100; i++)
  A[i] = ((B[i] * C[i]) + D[i]) / 2;
```

Part a

Translate this code into assembly language using the following instructions in the ISA (note the number of cycles each instruction takes is shown with each instruction):

Opcode	Operands	# Cycles	Description
`LEA`	`Ri, X`	1	Ri ← address of X
`LD`	`Ri, Rj, Rk`	11	Ri ← MEM[Rj + Rk]
`ST`	`Ri, Rj, Rk`	11	MEM[Rj + Rk] ← Ri
`MOVI`	`Ri, Imm`	1	Ri ← Imm
`MUL`	`Ri, Rj, Rk`	6	Ri ← Rj × Rk
`ADD`	`Ri, Rj, Rk`	4	Ri ← Rj + Rk
`ADD`	`Ri, Rj, Imm`	4	Ri ← Rj + Imm
`RSHFA`	`Ri, Rj, amount`	1	Ri ← RSHFA (Rj, amount)
`BRcc`	`X`	1	Branch to X based on condition codes

Assume it takes one memory location to store each element of the array. Also assume that there are 8 registers (R0-R7).

How many cycles does it take to execute the program?

(This is just one possible solution. There can be multiple solutions.)

```
        MOVI R0, 0          (1 cycle)
        LEA R4, A           (1 cycle)
        LEA R5, B           (1 cycle)
        LEA R6, C           (1 cycle)
        LEA R7, D           (1 cycle)
LOOP    LD R1, R5, R0       (11 cycles)
        LD R2, R6, R0       (11 cycles)
        LD R3, R7, R0       (11 cycles)
        MUL R1, R1, R2      (6 cycles)
        ADD R1, R1, R3      (4 cycles)
        RSHFA R1, R1, 1     (1 cycle)
        ST R1, R4, R0       (11 cycles)
        ADD R0, R0, 1       (4 cycles)
        ADD R1, R0, -100    (4 cycles)
        BNZ LOOP            (1 cycle)
```

5 × 1 + 100 × (11 + 11 + 11 + 6 + 4 + 1 + 11 + 4 + 4 + 1) = 6405 cycles

Part b

Now write Cray-like vector/assembly code to perform this operation in the shortest time possible. Assume that there are 8 vector registers and the length of each vector register is 64. Use the following instructions in the vector ISA:

Opcode	**Operands**	**Number of Cycles**	**Description**
`LD`	`Vst, #n`	1	Vst ← n
`LD`	`Vln, #n`	1	Vln ← n
`VLD`	`Vi, X + offset`	11, pipelined	
`VST`	`Vi, X + offset`	11, pipelined	
`Vmul`	`Vi, Vj, Vk`	6, pipelined	
`Vadd`	`Vi, Vj, Vk`	4, pipelined	
`Vrshfa`	`Vi, Vj, amount`	1	

How many cycles does it take to execute the program on the following processors? Assume that memory is 16-way interleaved.

Vector Processor (Solution 1):
The loop could be split into two parts as 36 and 64. Assume the vector code looks as follows:
(This solution assumes that the addressing mode VLD V0, B+36 exists - if it doesn't, then you would need 4 + 3 cycles using a pipelined adder to add 36 to A,B,C,D)
240 + 158 + 2 = 600

LD Vln, #36	(1 cycle)
LD Vst, #1	(1 cycle)
VLD V0, B	(11 + 35 cycles)
VLD V1, C	(11 + 35 cycles)
Vmul V2, V0, V1	(6 + 35 cycles)
VLD V3, D	(11 + 35 cycles)
Vadd V4, V2, V3	(4 + 35 cycles)
Vrshfa V5, V4, 1	(1 + 35 cycles)
VST V5, A	(11 + 35 cycles)
LD Vln, #64	(1 cycle)
VLD V0, B+36	(11 + 63 cycles)
VLD V1, C+36	(11 + 63 cycles)
Vmul V2, V0, V1	(6 + 63 cycles)
VLD V3, D+36	(11 + 63 cycles)
Vadd V4, V2, V3	(4 + 63 cycles)
Vrshfa V5, V4, 1	(1 + 63 cycles)
VST V5, A+36	(11 + 63 cycles)

i. Vector processor without chaining (vector instructions done serially)

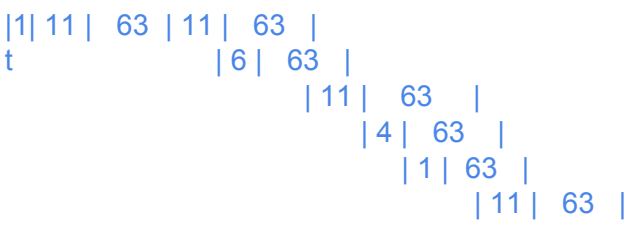
|1|1| 11 | 35 | 11 | 35 | 6 | 35 | 11 | 35 | 4 | 35 | 1 | 35 | 11 | 35 |
t

| 1| 11 | 63 | 11 | 63 | 6 | 63 | 11 | 63 | 4 | 63 | 1 | 63 | 11 | 63 |
t

2 + (46 × 4 + 41 + 39 + 36) + 1 + (74 × 4 + 69 + 67 + 64)= **799 cycles**

ii. Vector processor with chaining, 1 port to memory. Chaining means the machine begins the next operation as soon as the operands are ready.
w/ strip mining next load starts when first LD is done

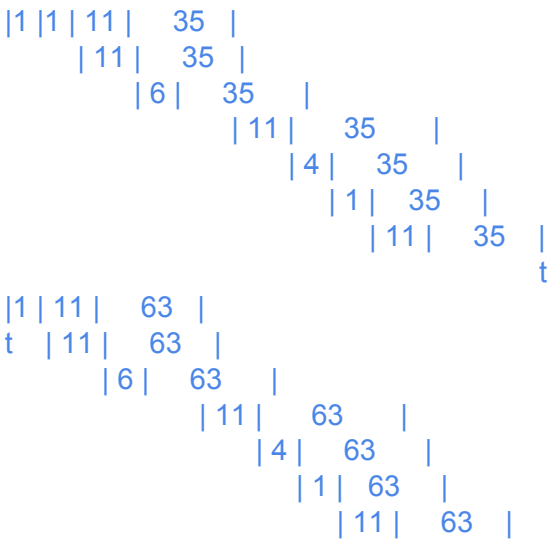
1	1	11	35	11	35
6	35
11	35
4	35
1	35
11	35
t



Chaining, in this instance, hides the VMULT, VADD, and VSHL operations. Memory becomes the primary bottleneck. Takes **483 cycles.**

Note: it is assumed here that you cannot change Vector length if prior operation using old Vector length is still executing.

iii. Vector processor with chaining; 2 loads, 1 store per cycle.



 # cycles = 275

Vector Processor (Another Solution)

Another solution is to split the loop into two equal parts as 50 and 50.

```
LD     Vln, #50          (1 cycle)
LD     Vst, #1           (1 cycle)
VLD    V0, B             (11 + 49 cycles)
VLD    V1, C             (11 + 49 cycles)
Vmul   V2, V0, V1          (6 + 49 cycles)
VLD    V3, D             (11 + 49 cycles)
Vadd   V4, V2, V3          (4 + 49 cycles)
Vrshfa V5, V4, 1           (1 + 49 cycles)
VST    V5, A             (11 + 49 cycles)
```

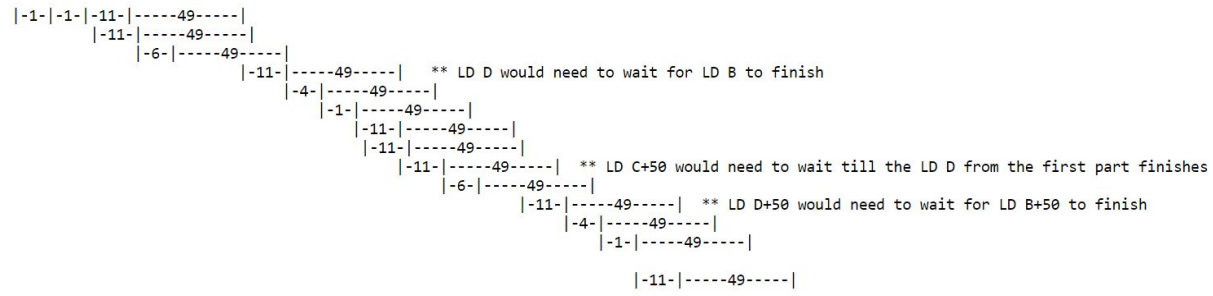
VLD V0, B+50 (11 + 49 cycles)
VLD V1, C+50 (11 + 49 cycles)
Vmul V2, V0, V1 (6 + 49 cycles)
VLD V3, D+50 (11 + 49 cycles)
Vadd V4, V2, V3 (4 + 49 cycles)
Vrshfa V5, V4, 1 (1 + 49 cycles)
VST V5, A+50 (11 + 49 cycles)

i. Vector processor without chaining (vector instructions done serially) similar to solution 1 minus one for vector length change operation

2 + (60 × 4 + 56 + 53 + 50) + (60 × 4 + 55 + 53 + 50) = **798 cycles**

ii. Vector processor with chaining, 1 port to memory. This part for solution will be the same as previous solution minus 1 for vector length change operation. Total = **482 cycles.**

iii.Vector processor with chaining; 2 loads, 1 store per cycle



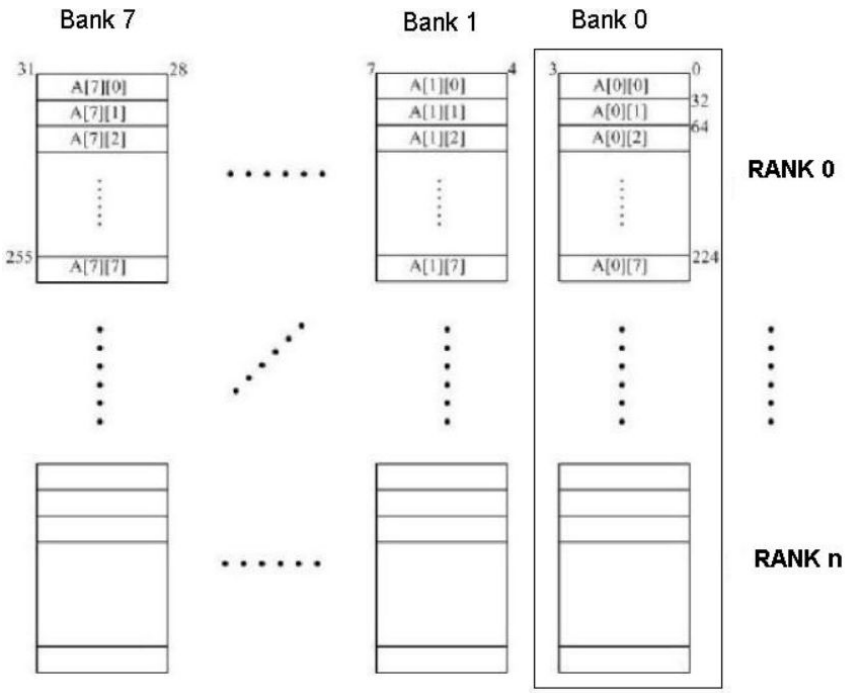
1 + 1 + (11 + 49) + 11 + 4 + 1 + 1 + (11 + 49) + 11 + 4 + 1 + (11 + 49) = **215 cycles**

Problem 2

Consider the following piece of code:

```
for(i = 0; i < 8; ++i){
  for(j = 0; j < 8; ++j){
    sum = sum + A[i][j];
  }
}
```

··The figure below shows an 8-way interleaved, byte-addressable memory. The total size of the memory is 4KB. The elements of the 2-dimensional array, A, are 4-bytes in length and are stored in the memory in column-major order (i.e., columns of A are stored in consecutive memory locations) as shown. The width of the bus is 32 bits, and each memory access takes 10 cycles.



A more detailed picture of the memory chips in Rank 0 of Bank 0 is shown below.

a. Since the address space of the memory is 4KB, 12 bits are needed to uniquely identify each memory location, i.e., Addr[11:0]. Specify which bits of the address will be used for:
    - Byte on bus - Addr[1:0]
    - Interleave bits - Addr[4:2]
    - Chip address - Addr[7:5]
    - Rank bits - Addr[11:8]

b. How many cycles are spent accessing memory during the execution of the above code? Compare this with the number of memory access cycles it would take if the memory were not interleaved (i.e., a single 4-byte wide memory chip).

577 Cycles. The first 8 memory accesses, A[0][0] to A[0][7], must occur sequentially with no overlap since they are all accesses to the same bank. Thus, it would take 80 cycles for the 1st 8 memory accesses, with the 8th access starting in cycle 70. Since the 8th and 9th memory accesses, A[0][7] and A[1][0], respectively, are to different banks, the accesses can overlap, and the 9th access can start in cycle 71 (70 cycles for the 1st 7 accesses plus 1 additional cycle of the 8th access). Continuing with this logic, the access to A[2][0] could start in cycle 142 (71x2). Finally, the access to A[7][0] could start in cycle 497 (71x7). Now all that remains are 8 more memory accesses, all to the same bank (A[7][0] to A[7][7]). This takes another 80 cycles, bringing the total to 577 cycles (497 + 80).

If the memory were not interleaved, all 64 memory accesses must happen sequentially with no overlap, so it would take a total of 640 cycles (64*10). Therefore, we do gain some benefit from this interleaving scheme, but not that much.

c. Can any change be made to the current interleaving scheme to optimize the number of cycles spent accessing memory? If yes, which bits of the address will be used to specify the byte on bus, interleaving, etc. (use the same format as in part a)? With the new interleaving scheme, how many cycles are spent accessing memory? Remember that the elements of A will still be stored in column-major order.

Yes, a change can be made. The new bits are:

Byte on bus Addr[1:0]
Interleave bits Addr[4:2]
Chip address Addr[11:9]
Rank bits Addr[8:5]

73 Cycles. With the new interleaving scheme, consecutive memory accesses are to either to different banks of the same rank, or to different ranks all together. In both cases, the consecutive accesses can start immediately after each other. Therefore the latency of all memory accesses would be hidden except the first access. Total number of cycles = 10 + 63 = 73

d. Using the original interleaving scheme, what small changes can be made to the piece of code to optimize the number of cycles spent accessing memory? How many cycles are spent accessing memory using the modified code?

Only one line of code needs to be changed:

sum = sum + A[i][j];
to
sum = sum + A[j][i];

Alternatively, you could keep that line the same, but swap the variable (i/j) of the inner and outer loops as shown below.

Original code:

```
for(i = 0; i < 8; ++i){
  for(j = 0; j < 8; ++j){
    sum = sum + A[i][j];
  }
 }
```

New code:

```
for(j = 0; j < 8; ++j){
  for(i = 0; i < 8; ++i){
    sum = sum + A[i][j];
  }
 }
```

87 Cycles. Now consecutive memory accesses are to different banks, so the accesses can overlap. The 1st access, A[0][0], would begin at cycle 0, the 2nd, A[1][0], at cycle 1, and so on. The 8th access, A[7][0], would start at cycle 7. However, the 9th access, A[0][1], cannot start at cycle 8. It would have to wait 2 more cycles for the 1st access to finish since it is on the same bank as the 1st access; therefore, it would start at cycle 10. Continuing this logic, the access to A[0][2] would start at cycle 20, and finally, the access to A[0][7] would start at cycle 70. Now, all that is left are 8 accesses, but they are all to different banks so they can start 1 cycle after each other. The access to A[1][7] would begin at cycle 71, A[2][7] at 72, and finally A[7][7], the last memory access, would begin at cycle 77 and, therefore, end at cycle 87

Problem 3

NOTE: The cycle numbers depend on your assumptions!!!! Write them down!!!

This question is about buses and refers to the system shown in the figure below (latencies are the numbers given). The processor performs the following list of accesses, which must be issued in the order below but can complete out of order.

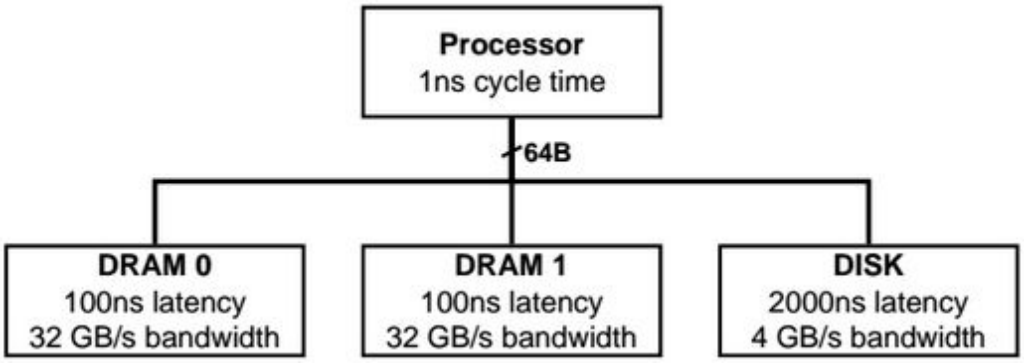
Assumptions:

- The latency of the wires is 0ns (1 bus traversal does not add latency).
- A request on the bus does not carry any data.
- The data is transferred in one long burst, and the time is determined by the bandwidth of each component. The entire transfer requires a single arbitration.

- Each component can only handle one access at a time and is busy until that access completes.
- There is no bus scheduling – all requests try to arbitrate only when they are ready and not before.
- If an arbiter is used, all transaction that lost must retry during the next arbitration cycle. Lower originating ID wins arbitration in case of a conflict and can start immediately following the arbitration delay.
- CSMA (collision sense multiple access) is a form of distributed arbitration, which requires retrying all transactions that attempt to use the bus at the same time (assume that in-progress transactions do not have to restart). The transmitter of data is considered the bus master.
- The command and data buses are arbitrated together; i.e. one device CANNOT send a command while another device sends data.

List of Accesses:

ID	Time (ns)	Component	R/W	Datasize
0	0	disk	R	4KB
1	100	DRAM0	R	32B
2	2000	DRAM1	R	32B
3	2050	DRAM1	R	32B
4	2100	disk	R	4KB
5	2110	DRAM1	R	32B



Fill in end time of each access in the table below, which describes different bus design options.

Bus Design	Time of Completion for access ID					
	0	1	2	3	4	5
Synchronous (10ns clock), pending transactions, centralized arbitration (one bus cycle (10ns) delay per arbitration).	3010	3130	3250	3370	6380	6500

Asynchronous, split transactions, CSMA arbitration (retries: lower access ID waits 1ns, higher access IDs all wait 5ns).	3001	201	3107	3208	6016	3309

There can be multiple solutions.

Assumptions:

1. Once bus is granted, the next arbitration cycle is known. So that as soon as data transfer is completed, transaction requests starts. So no cycle wasted in between.
2. Once bus is granted, the grant signal reaches component in 0 ns as latency is 0. As soon as grant signal is received, components starts its work (Latency starts).
3. At the end of latency time, data transfers begins.

Case 1: Synchronous and pending

1. ID 0: Disk
    a. Request at 0 ns
    b. Bus grant at 10 ns (1 cycle for arbitration)
    c. Latency of Disk = 2000 ns
    d. Time to transfer 4KB data = 4KB/4GBps = 1000 ns
    e. Time of completion = 3010 ns
2. ID 1: DRAM 0
    a. Requests at 3010 ns
    b. Bus grant at 3020 ns (1 cycle for arbitration)
    c. Latency of DRAM = 100 ns
    d. Time to transfer 32B data = 32B/32GBps = 1 ns
    Bus clock is 10ns, so even though transfer can happen in 1ns, it still takes 10ns before new requests can start
    e. Time of completion = 3130 ns
3. ID 2: DRAM 1
    a. Requests at 3010, 3130 ns
    b. Bus grant at 3140 ns (1 cycle for arbitration)
    c. Latency of DRAM = 100 ns
    d. Time to transfer 32B data = 32B/32GBps = 1 ns
    Bus clock is 10ns, so even though transfer can happen in 1ns, it still takes 10ns before new requests can start
    e. Time of completion = 3250 ns
4. ID 3: DRAM 1
    a. Requests at 3010, 3130, 3250 ns
    b. Bus grant at 3260 ns (1 cycle for arbitration)
    c. Latency of DRAM = 100 ns
    d. Time to transfer 32B data = 32B/32GBps = 1 ns
    Bus clock is 10ns, so even though transfer can happen in 1ns, it still takes 10ns before new requests can start
    e. Time of completion = 3370 ns

5. ID 4: Disk
    a. Request at 3010, 3130, 3250, 3370 ns
    b. Bus grant at 3380 ns (1 cycle for arbitration)
    c. Latency of Disk = 2000 ns
    d. Time to transfer 4KB data = 4KB/4GBps = 1000 ns
    e. Time of completion = 6380 ns
6. ID 5: DRAM 1
    a. Requests at 3010, 3130, 3250, 3370, 6380 ns
    b. Bus grant at 6390 ns (1 cycle for arbitration)
    c. Latency of DRAM = 100 ns
    d. Time to transfer 32B data = 32B/32GBps = 1 ns
    Bus clock is 10ns, so even though transfer can happen in 1ns, it still takes 10ns before new requests can start
    e. Time of completion = 6500 ns

Assumptions:

1. The ID's of only requesting accesses are considered for arbitration and not the ID which is currently using the bus.
   If transaction of ID -1 is in progress, and a new request for ID comes up, then ID request has to wait and keep trying at intervals of 1 ns till it finds that there is no collision. During this time if another requests comes with higher ID say ID + 1, then ID+1 request has to wait and keep trying at intervals of 5 ns till it finds that there is no collision. When ID starts its transaction, then wait time of ID+1 is reduced to 1ns as its the one with lower access id among arbitrating units
2. The change in waiting time from 5 ns to 1 ns happens at the time of trying for access the bus. So if at 3000ns, 3 ID's collide, then 1st ID waits for 1 ns and 2,3 ID wait for 5ns. At 3001 ns, ID 1 starts transaction. At 3005 both 2 and 3 check the bus at this as ID 2 is lower, its wait time becomes 1 ns.
3. In case of multiple requests to same component, the request is saved locally and is serviced after the current request completes

Case 2 Asynchronous and split

1. ID 0: Disk
    a. Uses bus at 0 ns to send request
    b. Data ready at 2000ns
    c. Tries to use bus at 2000ns, collision with ID2
    d. Wait time = 1ns
    e. Tries again at 2001 ns, gets bus
    f. Time to transfer 4KB data = 4KB/4GBps = 1000 ns
    g. Time of completion = 3001 ns
2. ID 1: DRAM 0
    a. Uses bus at 100 ns to send request
    b. Data ready at 200ns
    c. Tries to use bus at 200ns, no collision
    d. Time to transfer 32B data = 32B/32GBps = 1 ns
    e. Time of completion = 201 ns

3. ID 2: DRAM 1
    a. Sending requests every 5 ns since 2000 ns
    b. At 3005 collision detected with ID 3
    c. Wait time = 1ns
    d. Tries again at 3006 ns, gets bus
    e. Data from memory ready at 3106
    f. Tries to get bus at 3106, no collision
    g. Time to transfer 32B data = 32B/32GBps = 1 ns
    h. Time of completion = 3107 ns
4. ID 3: DRAM 1
    a. Sending requests every 5 ns since 2050 ns
    b. At 3010 collision detected with ID 4, 5
    c. Wait time = 1ns
    d. Tries again at 3011 ns, gets bus
    e. As DRAM 1 is already in use, the request goes into queue for DRAM 1
    f. DRAM 1 becomes ready for new access at 3107 ns
    g. Data ready at 3207 ns
    h. Tries to get bus at 3207, no collision
    i. Time to transfer 32B data = 32B/32GBps = 1 ns
    j. Time of completion = 3208 ns
5. ID 4: Disk
    a. Sending requests every 5 ns since 2100 ns
    b. At 3015 collision detected with ID 5
    c. Wait time = 1ns
    d. Tries again at 3016 ns, gets bus
    e. Data ready at 5016ns
    f. Tries to get bus at 5016, no collision
    g. Time to transfer 4KB data = 4KB/4GBps = 1000 ns
    h. Time of completion = 6016 ns
6. ID 5: DRAM 1
    a. Sending requests every 5 ns since 2110 ns
    b. At 3020, no collision, gets bus
    c. As DRAM 1 is already in use, the request goes into queue for DRAM 1
    d. DRAM 1 becomes ready for new access at 3208 ns
    e. Data ready at 3308 ns
    f. Tries to get bus at 3308, no collision
    g. Time to transfer 32B data = 32B/32GBps = 1 ns
    h. Time of completion = 3309 ns

Problem 4

If the latency of a DRAM memory bank is 37 cycles, into how many banks would you interleave this memory in order to fully hide this latency when making sequential memory accesses?

In order to fully hide the latency of Memory, we need atleast 37 banks.
As we want to keep the number of banks a power of 2, so as to directly assign particular bits in address for interleaving, the number of banks should be 64.