Indian Institute of Technology, Kharagpur
Mid-Spring Semester 2017-18

Date of Examination: 22-02-2018 Session: AN (2-4 pm) Duration: 2 hrs
Subject No.: CS31702
Subject: COMPUTER ARCHITECTURE AND OPERATING SYSTEMS
Department/Center/School: Computer Science and Engineering
Specific charts, graph paper, log book etc., required: NO Total Marks : 60
Special instructions (if any): ANSWER ALL QUESTIONS
Note: All parts of the question (a,b,c,d) should be answered at a stretch.

1. **(a) Name 8 great ideas exploited extensively by the designers of computer architectures.**

Design for Moores Law
Use abstraction to simplify design
Make the common case fast
Performance via parallelism
Performance via pipelining
Performance via prediction
Hierarchy of memories
Dependability via redundancy

(b) For the following MIPS instructions specify the instruction formats by splitting the 32-bit machine code into subsets and label them. Briefly name the above labels and interpret them in execution of the instruction.

i. Arithmetic instructions with register operands.

op: opcode (6 bits); rs: source reg-1 (5 bits); rt: source reg-2 (5 bits); rd: destination reg (5 bits); shamt: shift amount (5 bits); funct: function (6 bits)
opcode specify the broad class of operation (Ex: arithmetic).
function: specific operation within arithmetic (say add or sub).
Arithmatic instructions perform arithmetic operations on 2 source register operands (rs and rt) and the result will be stored at destination register (rd).
For arithmetic instructions the shmt field is zero.

ii. Arithmetic instructions with immediate operands.

op: opcode (6 bits); rs: source reg (5 bits); rt: destination (5 bits); constant: (16 bits)
opcode specify the arithmetic with immediate mode of operation.
These instructions perform arithmetic operations on source register operand (rs) and a constant. The result will be stored at destination register (rt).

iii. Memory access instructions

op: opcode (6 bits); rs: base reg (5 bits); rt: destination (5 bits); offset: (16 bits)
opcode specify the type of memory access operation (load or store)
These instructions first compute the target memory address by adding the offset to the contents of base register. Then it access the memory location either to read the value from the memory or write into memory. The second register present

in the instruction will be used to receive the value from memory in case of load or the value present in the register will be moved to memory in case of store.

iv. Branch instructions

op: opcode (6 bits); rs: reg-1 (5 bits); rt: reg-2 (5 bits); label: (16 bits)
opcode specify the type of branch operation (bne or be)
These instructions first check whether the contents present in two registers are equal or not. Based on the validity of the condition, the branch decision will be made. If branch is taken then the value present in the label will be multiplied by 4 and add to pc+4. Otherwise, if branch is not taken the following instruction (pc+4) will be executed.

v. Jump instructions

op: opcode (6 bits); address: (26 bits);
opcode specify that it is jump operation (unconditional)
This can be viewed as an unconditional jump instruction. The target address is calculated as follows: 4 MSBs of pc+4 address concatenated with 2-bit left shifted address present in the instruction.

(c) Write the MIPS code for the following C function. Provide suitable comments against to each MIPS instruction for understanding its functionality.

```
int sum(int a, int b, int c, int d)
{
int e;
e = a+b+c+d;
return(e);
}
```

Before calling the function, the function arguments (parameters) are stored in registers a0, a1, a2 and a3. The contents of PC+4 (address of next instruction to be executed) are stored on register ra (for returning to the instruction where the functional call has taken). Use jal instruction to move the control to procedure.

```
sum :
addi sp, sp, -4 (Pushing the sp (stack pointer) down to create a space for storing the contents of save reg)
sw s0, 0(sp) (storing the contents of s0 on stack. Because we need to use s0 register in the procedure call for storing the sum.)
add t0, a0, a1 (adding first two arguments of a function and place the result in temporary register t0)
add t1, a2, a3 (adding last two arguments of a function and place the result in temporary register t1)
add s0, t0, t1 (computing the final sum and stored in s0)
add v0, s0, $0 (move the contents of s0 to v0)
lw s0, 0(sp) (restore the earlier contents of s0 from sp)
addi sp, sp, 4 (push the sp up to remove the s0 reg)
jr ra (return the control to an address mentioned in register ra)
```

(d) Answer the following in view of IEEE 754 Single precesion floating point number representation:

i. Representation format with appropriate details (number of bits and labels associated to different subsets).
Representation Format: sign: 1 bit, exponent: 8 bits and fraction: 23 bits

ii. How the floating point value is computed from the above representation?
$(-1)^S \times (1 + Fraction) \times 2^{(Exponent-bias)}$

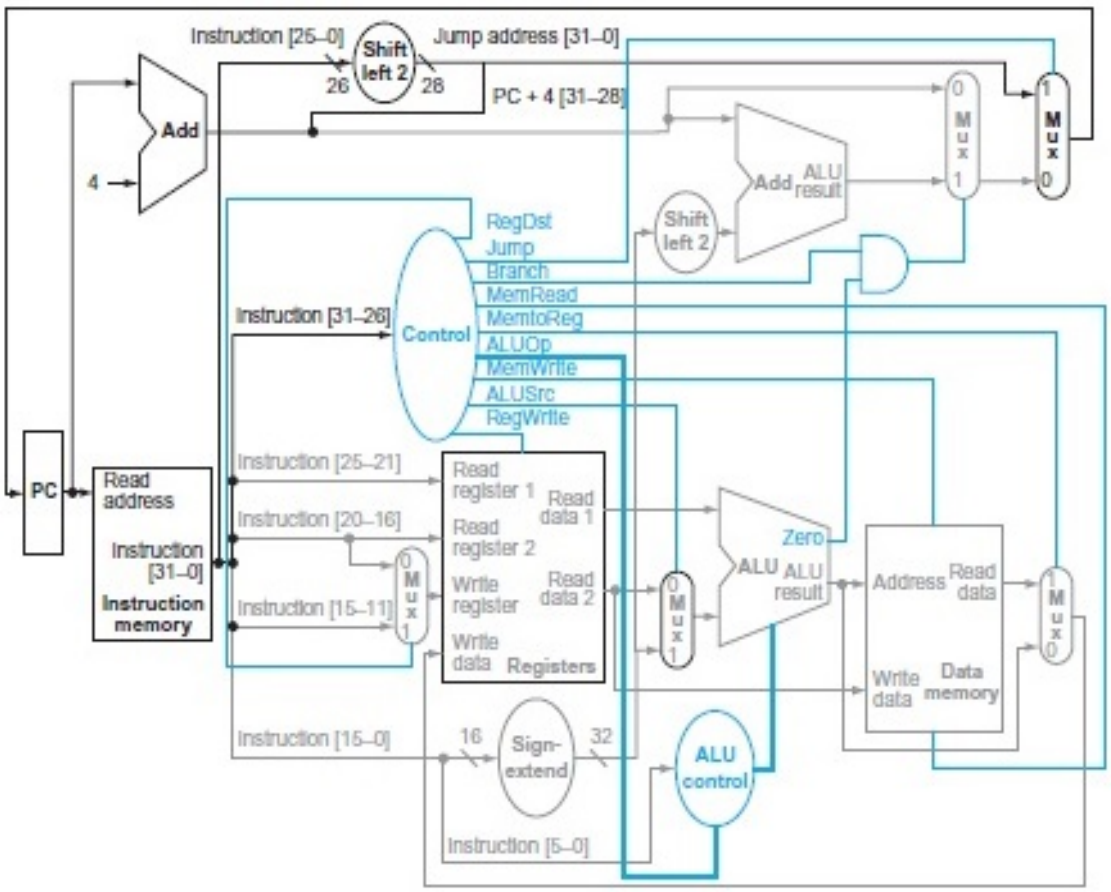
iii. Compute the range of numbers that can be represented using single precesion encoding.
$-1.11111111111111111111111 \times 2^{-126}$
$+\,1.11111111111111111111111 \times 2^{+127}$

(2+10+5+3 = 20M)

2. (a) Discuss how combinational and sequential logics differ? Provide 2 example circuits to each of the logic.
Combinational logic: Output depends only on current inputs. Examples: ALU, Multiplexer, gates, controller
Sequential logic: It has atleast 2 inputs and one output. The inputs are data and clock. Clock determines when the data to be written into the element. Output of sequential logic depends on the inputs as well as internal state of the element. Examples: memory, registers, flipflops, ...

(b) Design data path and controller for a single-cycle processor to execute R-type, memory reference and control flow instructions. Draw a neat sketch with all hardware components and place the connections and signals carefully. Highlight the salient points in the design of above mentioned datapath and controller.

In single cycle processor each resource will be available through out the cycle for executing the instruction. But, each resource (functional unit) can be used only once during execution of an instruction. With these constraints, a single cycle processor consists of 2 memory units: one for holding the instructions known as instruction memory used during fetching the instruction from the memory and other one used to hold data known as data-memory mainly meant for data access from memory (load and store instructions). Initially the entire program (all instructions) is in instruction memory. For fetching the instructions in sequence (one-by-one), program counter contains the address of the present instruction to be fetched from the memory. At the end of the execution of this instruction, the next instruction to be executed will depend on the type of the present instruction. If the instruction is branch instruction, the branch target address depends on the condition checked between the operands present in two registers. In case of branch-on-equal instruction, if the contents of two registers are same, then the target address is sum of the offset present in the instruction and PC+4. If branch is not successful, then the next instruction to be fetched is PC+4. For executing the branch instruction we need a controller which decode its opcode and recognize it as branch instruction, and up on knowing it as branch, we need to compute both PC+4 as well as PC+4+offset.



Parallelly, we need to check the contents of registers, whether they are same or not by using ALU functionality. For calculating PC+4 and PC+4+offset, we need 2 more adders. So for executing branch instruction, we need 2 adders, 1 ALU and a 2-to-1 multiplexer to choose the address between PC+4 and PC+4+offset based on the validity of the condition. To accommodate jump (unconditional branch) instruction, after fetching while decoding the instruction the controller come to know that its jump instruction and hence the address of the next instruction to be fetched will be computed by concatenating the 4 MSBs of PC+4 and 2-bit left-shifted 26 LSBs of JMP instruction. To include this instruction along with branch and other R-type and memory-access instructions, we need to include one more 2-to-1 multiplexer, where the 2 inputs are jump-target address and PC+4 or PC+4+offset (branch). For executing R-type instructions we need instruction memory (instruction memory is required for all types of instructions), register-file, controller and ALU. In R-type instruction source and destination operands are from the registers. Based on the type of arithmetic operation, the ALU has to be controlled for specific functionality. For the considered 3 types of instructions (R-type, memory-access and control flow) the 2 inputs of ALU may come from registers or one from register and other from the offset present (lw, sw) in the instruction itself. Therefore for selecting the 2nd input (either from register or from instruction offset) a 2-to-1 multiplexer is employed. In R-type instruction, the result produced by ALU will be stored back in one of the registers of the reg-file. Even for lw instruction also, the contents of specific memory location will fetched and written into one of the registers of the reg-file. The registers that were used to store the result in case of R-type and memory content in case

of lw are denoted by 2-different sets of bits in the instruction. Therefore to select the specific set of bits from 2 sets, to choose the destination register for storing the result or loading the memory content, we need one more 2-to-1 multiplexer. After specifying the register identity, the controller should enable the write signal to reg-file to write the desired contents into specific register. In case of load/store instructions ALU will be used for memory address computation with adder functionality with one input from the register and other from the offset present in the instruction. Before addition, the offset has to shift left by 2 bits to convert word address to byte address. The memory address generated by ALU will be used by lw/sw to fetch/store the data from the data memory. For reading/writing the data into data memory along with the address, the controller should generate read/write signals based on the desired operation as per the instruction. In both R-type and load instructions, the result (either from arithmetic/logical operation or from memory content) will be stored back into one of the registers of the reg-file. Therefore, one should select the output from 2 possibilities (either the output of ALU or output of data memory) based on the type of instruction (R-type vs lw), we need to add 2-to-1 multiplexer for chosing the appropriate one based on the type of instruction.

For guiding the above mentioned datapath (various functional units) for executing various types of instructions, we need to design a controller circuit which will generate appropriate control signals to activate the functional units (datapath) as per the desired instruction. In this limited scenario, a single cycle processor which can support R-type, Memory-access and flow control instructions needs a controller with 10 control signals. These control signals consists of (i) signals for multiplexers (includes jump and branch), (ii) memory read, (iii) Memory write, (iv) reg write and (v) ALU control. Five control signals for five 2-to-1 multiplexers, three signals for each of MR, MW and RW operations and 2 signals for ALU operation which specify (memory:00, branch:01 and R-type:10) the broader group of instruction and specific ALU function will be specified by 3 bits (AND:000, OR:001, ADD: 010, SUB: 110 and SLT: 111).

(c) How a multi-cycle processor (without pipelining) differ from single-cycle processor? Mention its advantages as well as disadvantages over single-cycle processor?

In multi-cycle processor each instuction requires multiple clock cycles for its execution. The design is based on the principle that each operation (such as fetching, decoding and reading the register contents, ALU functionality and memory access) will be carried out in one clock cycle. Based on this different instructions will consume different nmber of cycles based the type of instruction.

Where as in single cycle processor, the design principle is that each instruction will complete its execution in one clock cycle. In this scenario, the duration of clock cycle to be set for the worst case, where the duration required for a critical/complex instruction. Therefore the efficiency is not optimal.

Advantages of multi-cycle processor: (i) Resource optimization, (ii) Overall CPU time for program execution is better compared to single cycle processor

Disadvantages of multi-cycle processor: (i) Additional hardware (registers and multiplexers) and (ii) More complex controller.

(d) Mention the additional functional units (hardware) and modifications in the datapath of a 5-stage multi-cycle pipeline processor (assume there

is no need to address hazards), when compared with the datapath of single-cycle processor in executing R-type, memory reference and control flow instructions.

(i) pc+4 has to be fed-back to pc soon after the 1st cycle.
(ii) pipeline registers in-between the stages
(iii) Destination register identities need to be passed across the stages in view of writing into register at stage-5 (example: lw instruction).
In case of single cycle processor any instruction will complete its execution in one clock cycle. Hence while designing the clock, the duration of the clock cycle (or clock frequency) is set such that the most complex (i.e., lengthy instruction) instruction can be completed within a clock cycle. Where as in multi-cycle pipelined processor each instruction execution takes multiple cycles based on the type of instruction. In each cycle one new instruction will be fed into the pipeline. Based on the length of the pipeline (number of stages) the number of instructions symultaneously executed is equal to number of stages. In each cycle each instruction will complete one of the stages of the pipeline and at the beginning of the new cycle instructions will move to the next/following stage. Since, parallelly several instructions are executing simultaneously to keep track of the information related to instructions at the end of the cycle the status of each instruction (in the respective pipeline stage) will be stored in the inter-stage pipeline register, which will be used in the following cycle. For each new cycle, a fresh/new instruction needs to be fetched from the memory for execution. Therefore, the PC has to be updated with PC+4 in each cycle for fetching the new instruction. These inter-stage pipeline registers are also used for holding the control signals in an appropriate manner and passing them to the following stages at suitable clock cycles. In 5-stage pipeline, for every instruction, all its control signals are generated in its 2nd stage. But, they have to be supplied/asserted at different times (clock cycles) for different stages, based on the movement of the instruction across the stages. For this purpose inter-stage pipeline registers are used to hold the contol signals and release them appropriately.

(2+8+2+3 = 15M)

3. **(a) Discuss the core reasons for structural, data and contorl hazards in case of pipeline processors. Suggest the strategies to minimise the effect of the above hazards to improve the efficiency of pipeline processor. Provide more details in case of compensating the data and control hazards.**

(i) Structural Hazard: Multiple instructions inside the pipeline needs specific hardware at the same time. Example: Single memory contains both instructions and data. In this case instruction fetching will conflict with data access with memory. This can be overcome by replicating the hardware/resources (splitting the memory into 2 units: instruction memory and data memory)
(ii) Data Hazards: Due to dependence of one instruction on earlier one which is still in pipeline. One way to overcome these hazards is to stall the pipeline at a specific cycle where one of the instruction expects the input from the earlier instruction, which is not available at that time. By stalling the pipeline for 1 or 2 cycles, the desired input from the earlier instruction will be available for processing. In some cases data-forwarding techniques will resolve data hazards without stalling the pipeline. Another possible solution to avoid data hazards is to reorder the code by a compiler.

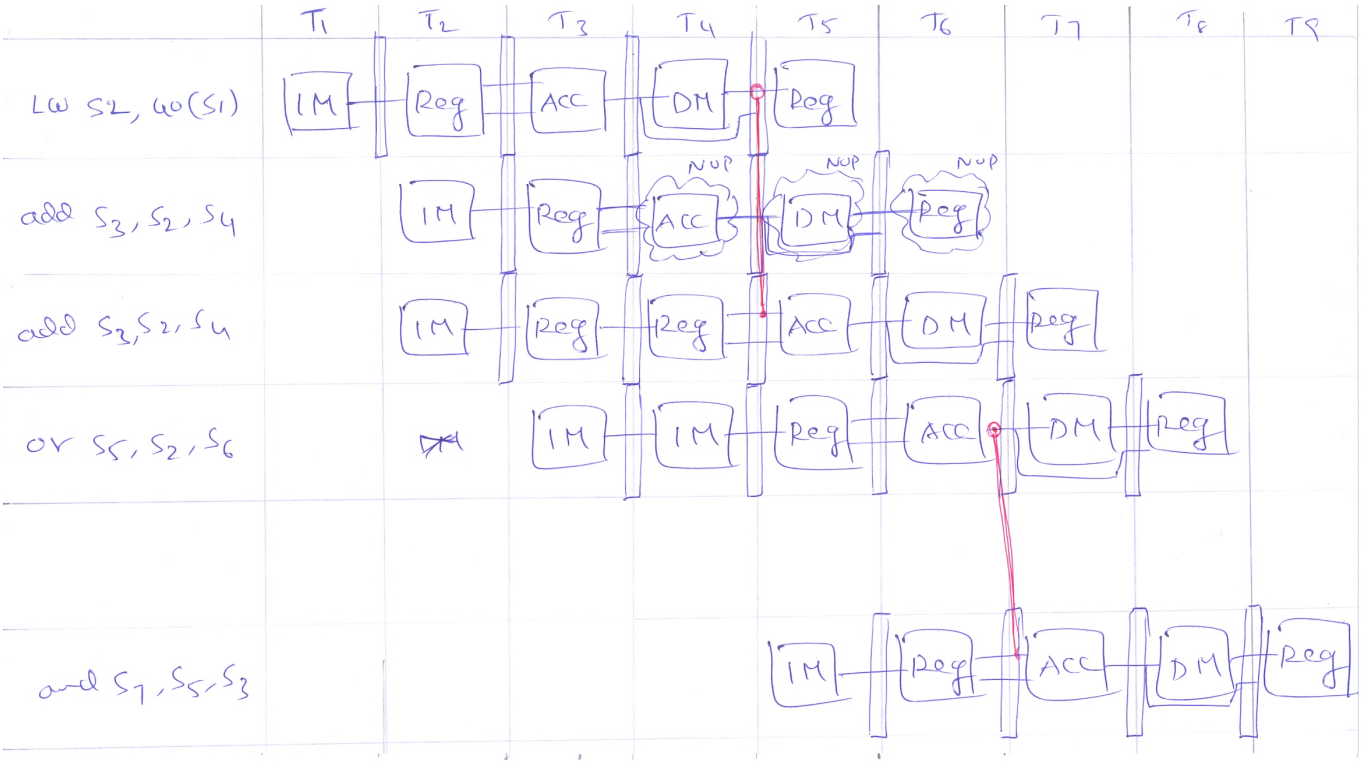
(iii) Control Hazards: These hazards will be observed in case of execution of branch instructions. In branch instructions, branching will be decided based on the condition to be checked. By that time the following instructions enter into the pipeline, and hazards will result if branch is successful. In general, the control hazards will come to know in the 4th stage, by that time 3 following instructions have already entered into the pipeline. In case if branch is successful, we need to flush those 3 instructions and pc has to be updated with branch target address. With this 2-3 CPU cycles will be wasted. The affect of branch-success can be minimized if we carry out condition check and branch target address computations in 2nd stage, we can reduce the wastage of CPU cycles to 1. If we want to improve the performance further in presence of control hazards, dynamic and static branch predictions may be explored. There exists another solution known to be delayed branch. In delayed branch, after the branch instruction place few instructions which are not affected by branch. With this we need not flush the instructions and CPU cycles are not wasted.

(b) Illustrate the flow of the following sequence of instructions through 5-stage multi-cycle pipeline processor using suitable diagrams synchronized to clock.

lw s2, 40(s1)
add s3, s2, s4
or s5, s2, s6
and s7, s5, s3

Clearly show and explain (in sequence w.r.t clock) the sequence of events (hazards) occur and actions taken against to the hazards in view of execution of the above sequence of instructions.

The lw instruction will be executed smoothly and complete in 5 cycles. The add



instruction following the lw needs the contents of s2 in its 3rd stage. But, the

contents of s2 will be updated by lw instruction in its 5th stage (i.e., 4th stage of add instruction). Since the earlier instruction to add is lw, only at the end of 4th stage of lw (end of 3rd stage of add) can provide the contents of s2. Therefore, in this scenario data forwarding is not applicable, and the only option is to stall the execution stage of add instruction for one cycle. The details of stalling an instruction at specific stage are discussed below. Here we are stalling the add instruction at 3rd stage (EX). It means the instructions in the preceding 2 stages have to be preserved and NOP (no operation) has to be introduced in EX stage of add instruction. Later this NOP bubble will propagate through the following stages in successive clock cycles. In stage-2, ID of add will take place and stage-1 the following instruction or is fetched from memory. During NOP at stage-3, as we are preserving the contents of inter-stage registers and pc, the same activity (in the earlier cycle) is repeated during NOP. So for every insersion of NOP, one CPY cycle will be wasted.
In the next cycle (i.e., 4th stage of lw), lw will fetch the value from memory and stored in pipeline register at the end of the clock cycle and further written into s2 register in stage-5. At the begining of stage-5 (stage-4 of add) of lw, the contents present in the pipeline register are forwarded to EX stage of add. However, or instruction executes smoothly without any hazard, after the NOP. During NOP, or instructon is in IF stage and it is in the same stage for 2 cycles due to NOP.
for the 4th instruction and, it encounters data hazard due to the dependency on its earlier instruction or. However this dependency can be overcome by data-forwarding from EX stage of or instruction to EX stage of and instruction.

(c) How the pipeline stalls will be implemented? *Pipeline need to be stalled in case of data hazards if data forwarding is not applicable (situation where forwarding in -ve time). For example arithmetic instructions followed by load where arithmetic operation has to be performed on the register contents where the register has to be loaded from the memory. In this case, the execution stage of arithmetic instruction has to be stalled for cycle. During that cycle the pipeline stages preceding to EX stage i.e., ID and IF stages have to be preserved and control signals to EX stage to be deasserted (make them zero). which means no-operation bubble has been created at EX stage and it has to propagate through following stages in successive cycles by nullifying the control signals of the respective stages in respective clock cycles. Preserving the preceeding stages means the contents of PC and inter-stage pipeline register IF/ID should not be changed in the following cycle. With this effect, during NOP the preceding stages will repeat the same action as in their earlier cycle.*

(5+8+2 = 15M)

4. (a) Briefly explain the following (in the context of memory access through memory hierarchy):

i. Compulsory miss & its solution
At the initial time of program execution, when entire cache is empty the first few instructions through a miss. This is invitable (no option) and compulsory. These misses can be reduced by increasing the block size so that based on the principle of spacial locality, the further conpulsory misses will be reduced.

ii. Capacity miss & its solution
As the cache size is limited and much smaller than the physical memory, capacity

misses will result. One can reduce the capacity misses by increasing the size of the cache.

iii. Conflict miss & its solution

There are also known to be collission misses. If a cache entry can hold very few blocks, and multiple entries of physical memory are mapped to single or few entries of cache, then the collision/conflict arises during replacement. We can minimise these misses either by increasing the cache size or by increasing the associativity between physical and cache memories.

iv. Write-through policy

Suppose, if CPU needs to write into data cache, then the copy of the data present in the physical memory also need to be changed. In write-through policy whenever you are writing into cache, at the same time write into main memory also. But, accessing main memory takes 100 CPU cycles, where CPU has to wait for long time. Therefore, instead of writing into main memory immediately, CPU first write into buffer, later it will be transfered to main memory.

v. Write-back policy

Suppose, if CPU needs to write into data cache, then the copy of the data present in the physical memory also need to be changed. In write-back policy whenever you are writing into cache, set the dirty-bit to 1, and CPU will continue in execution of the following instructions. The update in the main memory will takes place at the time of replacement of the block. At the time of block replacement, if dirty-bit is set, the block to be replaced is first write into the memory and it is replaced with new one. Here also to increase the performance of CPU, write buffer may be used for immediate purpose.

(b) Design a set-associative cache for the following specifications: (i) size of main memory = 16 MB, (ii) size of cache (only data) = 64 KB, (iii) block size = 64 bytes and (iv) number of sets = 512. Here, CPU access data at word level. Show the hardware implementation of the cache for the above specifications. Mention the following details in view of the above set-associative cache:

i. Size of the Tag field

Size of the physical address = 24 bits
Number of bits for representing set index = 9 bits
Amount of data stored in each set = $\frac{64KB}{512}$ = 128 bytes (7 bits)
Tag size = 24 - 9 - 7 = 8 bits

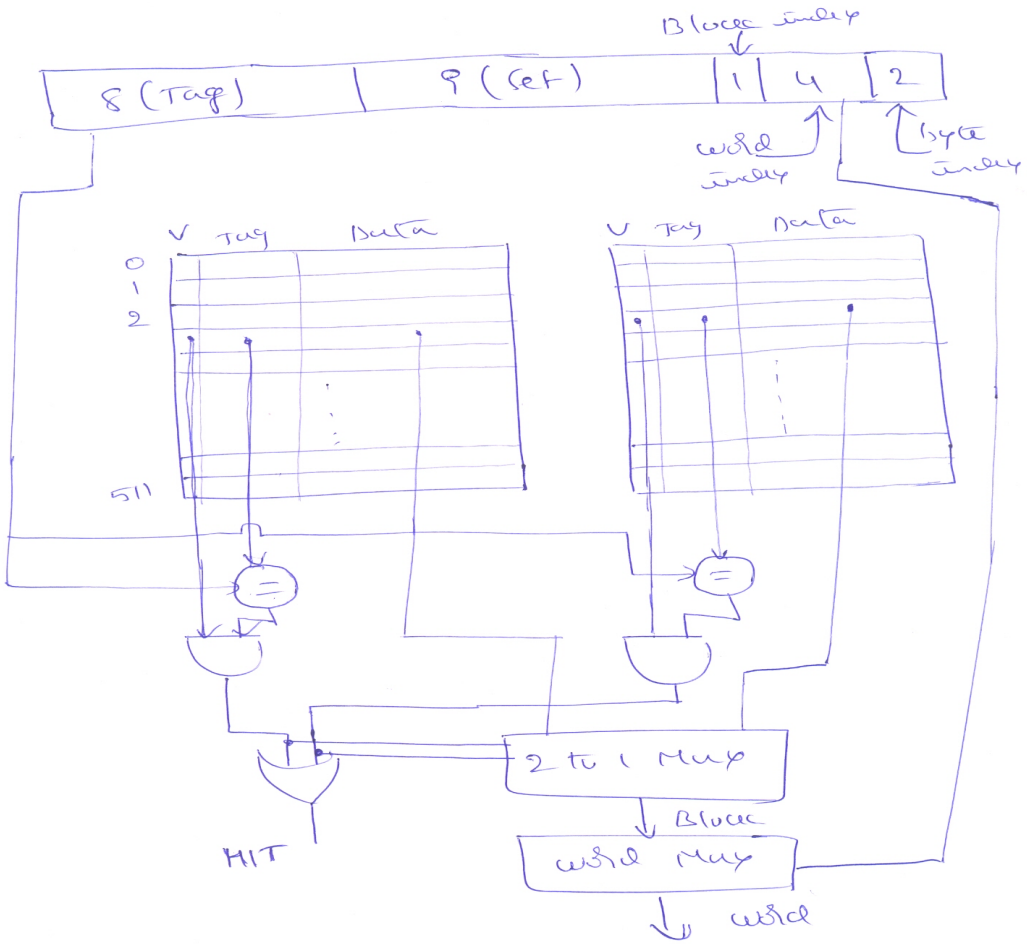
ii. In the context of n-way associative, what is the value of "n" in this problem?

Number of blocks in a set = $\frac{128}{64}$ = 2. Hence the given problem is 2-way set associative cache memory.

iii. What is the total size of the cache including the overhead?

Total size of the cache = size of valid bits + size of tag bits + data = 1024 bits + 1024 bytes + 64 KB = 65.125 KB

(5+5 = 10M)



Size of Physical address = 24 bits ($16\,MB = 2^{24}$ bytes)

Data size in each ~~cache~~ set = $\frac{64\,KB}{512} = 128$ bytes

blocks within a set = $\frac{128}{64} = 2 \Rightarrow$ 2-way set associative

bits for rep set index = $\log_2 512 = 9$ bits

bits for ~~w~~ Tag = $24 - 9 - 7 = 8$ bits

Cache overhead = ~~to~~ $2 \times 512 \times (1+8) = 1.125\,KB$

Total cache size = $64 + 1.125 = 65.125\,KB$