REG.NO.:

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF ELECTRONICS ENGINEERING
CONTINUOUS ASSESSMENT TEST - 2
WINTER SEMESTER 2025-2026

SLOT: D1+TD1

Programme Name & Branch : **B.Tech. & VLSI**
Course Code and Course Name : **BEVD207L - Computer Architecture**
Faculty Name(s) : **Arunachalam V, DHANABAL R**
Class Number(s) : **VL2025260500891, 0889**
Date of Examination : **18-March-2026**
Exam Duration : **90 minutes** **Maximum Marks: 50**

General instruction(s):

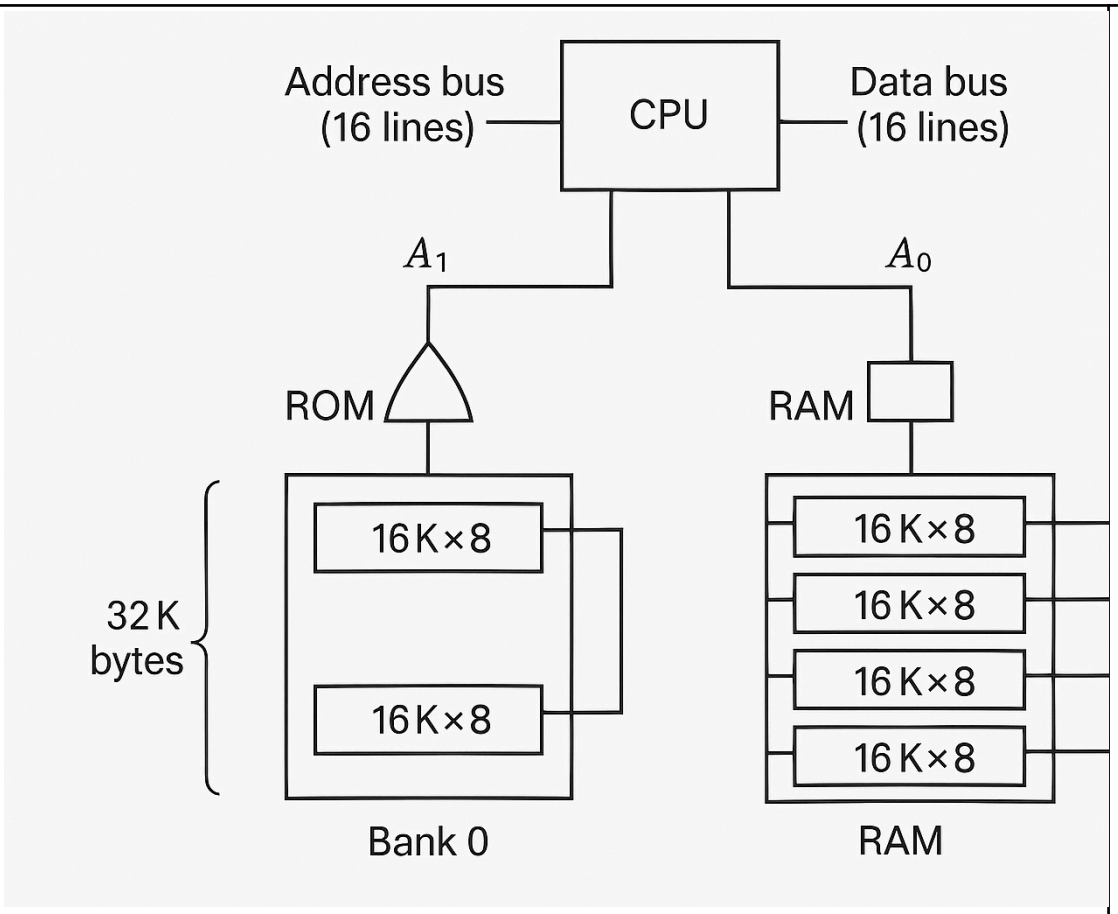
- Answer All Questions
- M - Max mark; CO – Course Outcome; BL – Blooms Taxonomy Level (1 – Remember, 2 – Understand, 3 – Apply, 4 – Analyse, 5 – Evaluate, 6 – Create)
- Course Outcomes

3. Understand the various memory systems.

4. Understand the processor design control unit.

Q. No	Question	M	CO	BL
1.	A computer system has the following cache memory parameters: Cache size: 64 KB, Block size: 16 bytes, Hit time: 4 cycles, Miss penalty: 70 cycles, Total memory accesses: 14,000, Cache hit rate: 85%. a. How many blocks are there in the cache? b. How many cache misses occur? c. How much time is spent on cache hits? d. How much time is spent on cache misses? e. What is the average memory access time (AMAT)? Anskey: **Given:** • Cache size = **64 KB = 65,536 bytes** • Block size = **16 bytes** • Hit time = **4 cycles** • Miss penalty = **70 cycles** • Total memory accesses = **14,000** • Cache hit rate = **85% = 0.85** **(a) Number of blocks in the cache** $$Number\ of\ blocks = \frac{Cache\ size}{Block\ size}$$ $$= \frac{65{,}536}{16} = 4096$$ ✅ **Number of blocks = 4096** --- **(b) Number of cache misses** $$Miss\ rate = 1 - 0.85 = 0.15$$ $$Cache\ misses = 14{,}000 \times 0.15 = 2{,}100$$ ✅ **Number of misses = 2,100** --- **(c) Time spent on cache hits**	10	3	2

	$Hits = 14,000 \times 0.85 = 11,900$ $Hit\ time\ total = 11,900 \times 4 = 47,600\ cycles$ ✅ **Time spent on hits = 47,600 cycles** **(d) Time spent on cache misses** Time per miss includes only the miss penalty. $Miss\ time\ total = 2,100 \times 70 = 147,000\ cycles$ ✅ **Time spent on misses = 147,000 cycles** **(e) Average Memory Access Time (AMAT)** $AMAT = Hit\ time + (Miss\ rate \times Miss\ penalty)$ $= 4 + (0.15 \times 70)$ $= 4 + 10.5 = 14.5\ cycles$ ✅ **AMAT = 14.5 cycles** **Final Answers Summary** **Part** / **Result** a. Blocks in cache — **4096** b. Cache misses — **2100** c. Time on hits — **47,600 cycles** d. Time on misses — **147,000 cycles** e. AMAT — **14.5 cycles**			
2.	Design a computer system with a processor having 16-bit address bus and 16-bit data bus. The system has a total of 128 kB of memory interfaced with the processor. Out of these 128 kB, 32kB is ROM and 96 kB is RAM. Individual memory chips are of capacity 16k x 8 bits. Thus, each 16-bit memory word has been configured by connecting two such memory chips in parallel. a. Find the number of Address lines to be used to access all parts of memory. b. Determine the total number of memory chips required and the number of chips per bank. c. Calculate the number of memory banks required. d. Draw the memory organization. e. Provide the addressing scheme used by the processor. Answer key:	10	3	3



Below is a complete design and the answers to each part.

Assumption (fits the numbers): the processor is **word-addressed** (16-bit word). With a 16-bit address bus it can address $2^{16} = 65,536$words = **128 KiB** (since each word is 2 bytes).

Given

- Address bus: **16 bits** (word addresses)
- Data bus: **16 bits**
- Total memory: **128 KiB = 65,536 words**
  - ROM: **32 KiB** (= 16K words)
  - RAM: **96 KiB** (= 48K words)
- Memory chips: **16K × 8 bits** (i.e., 16K byte-wide locations per chip)
- Each 16-bit word is formed by **two 8-bit chips in parallel** (lower byte + upper byte)

(a) Number of address lines used

- Each **16K×8** chip needs $log_2(16\,K) = 14$address lines to select 16K locations.
- We organize memory into **banks** of **16K words = 32 KiB** (because two 16K×8 chips in parallel give 16K×16).
- There are **4 banks** (see parts b/c). Selecting one of 4 banks needs **2** high-order address lines.

$$Total\ address\ lines\ used = 14\ within\ chip) + 2\ (bank\ select) = 16$$

These are exactly the processor's A15..A0.

(b) Total number of memory chips and chips per bank

- One **bank** (16K words = 32 KiB) requires **2 chips** (one for D7..D0, one for D15..D8).
- Total memory = **128 KiB = 4 banks** ⇒ $4\ banks \times 2\ chips/bank = 8\ chips\ total$.
- **Chips per bank:** 2.

Breakdown:

- **ROM 32 KiB:** 1 bank ⇒ **2 chips** (lower + upper byte), **/WE** tied inactive.
- **RAM 96 KiB:** 3 banks ⇒ **6 chips**, **/WE** driven by CPU write control.

(c) Number of memory banks required

Each bank = **32 KiB** ⇒ $128\ KiB/32\ KiB = 4\ banks$.

Allocation: **Bank 0 = ROM**, **Banks 1–3 = RAM**.

(d) Memory organization (block diagram)

I've generated a clean block diagram showing the organization (CPU, 2-to-4 bank decoder from A15..A14, parallel 16K×8 chips per bank, data-bus split to upper/lower byte, and control routing).

Textual wiring description (for reference):

- **Address bus (A15..A0):**
  - **A13..A0** → tied in parallel to the address inputs of **all chips** (14 lines used by each chip).
  - **A15..A14** → 2-to-4 decoder → generates **CS_B0 … CS_B3** (one per bank).
- **Data bus (D15..D0):**
  - **D7..D0** ↔ lower-byte chip of the selected bank.
  - **D15..D8** ↔ upper-byte chip of the selected bank.
- **Control lines:**
  - **/RD** → **/OE** of the two chips in the selected bank.
  - **/WR** → **/WE** of the two **RAM** chips in the selected bank (ROM **/WE** tied inactive).
  - (Optional) If the CPU supports byte writes, gate **/WE** to upper/lower chips with **/UB** and **/LB** byte-enable signals; otherwise both bytes are written per word.

(e) Addressing scheme used by the processor

Since the processor is **word-addressed**, each address selects a **16-bit word**.

Let **A[15:14]** select the bank, and **A[13:0]** be the word offset inside the bank.

Bank select (using A15..A14)

A15 A14	Bank	Device	Size
0 0	**Bank 0**	**ROM**	32 KiB (16K words)
0 1	**Bank 1**	**RAM**	32 KiB (16K words)
1 0	**Bank 2**	**RAM**	32 KiB (16K words)
1 1	**Bank 3**	**RAM**	32 KiB (16K words)

Word-address map (what appears on A15..A0)

- **ROM (Bank 0):** 0x0000 – 0x3FFF (16K words)
- **RAM (Bank 1):** 0x4000 – 0x7FFF
- **RAM (Bank 2):** 0x8000 – 0xBFFF
- **RAM (Bank 3):** 0xC000 – 0xFFFF

Equivalent byte-address ranges (if you prefer bytes)

Multiply word addresses by 2 (LSB byte offset is implicit):

- **ROM:** 0x00000 – 0x07FFF (32 KiB)

	• **RAM:** 0x08000 – 0x0FFFF (Bank 1) **RAM:** 0x10000 – 0x17FFF (Bank 2) **RAM:** 0x18000 – 0x1FFFF (Bank 3) Key **(a)** 16address lines (A15..A0): 14 to chips, 2 for bank select. **(b)** 8 *chips total*; 2 *chips per bank*. **(c)** 4 *banks*(1 ROM, 3 RAM). **(d)** Diagram provided above + wiring description. **(e)** Addressing scheme and maps given (banked by A15..A14).			
3.	a. Consider a demand paging memory management system with 32-bit logical address, 20 -bit physical address, and page size of 2048 bytes. Assuming that the memory is byte addressable, what is the maximum number of entries in the page table? b. Find the physical address for the logical address 2010 in a system uses paging with the logical address space = 16 bits, page size = 1 KB, and Page table: Page No | Frame No 0 | 5 1 | 6 2 | 3 3 | 7 Key: **a. Maximum number of entries in the page table** **Given** • Logical address = **32 bits** • Page size = **2048 bytes = 2 KB** • Memory is **byte-addressable** **Step 1: Determine the number of offset bits** $Page\ size = 2048 = 2^{11}$ So offset = **11 bits** **Step 2: Determine number of page number bits** $Page\ number\ bits = 32 - 11 = 21$ --- **Step 3: Maximum number of page table entries** $Number\ of\ pages = 2^{21}$ **Final Answer for (a):** $2^{21} = 2,097,152\ page\ table\ entries$ **b. Physical address for logical address 2010** **Given** • Logical address space = **16 bits** • Page size = **1 KB = 1024 bytes** • Page table: **Page No** | **Frame No** 0 | 5 1 | 6	10	3	2

	2 3 3 7 **Step 1: Compute offset and page number** 1 KB = 1024 = 2^{10} → Offset = **lower 10 bits** $Logical\ address = 2010_{10}$ Compute page number: $Page\ number = \lfloor \frac{2010}{1024} \rfloor = 1$ Compute offset: $Offset = 2010 mod 1024 = 986$ --- **Step 2: Use page table to find frame** Page 1 → Frame 6 --- **Step 3: Compute physical address** $Physical\ address = (frame\ no \times frame\ size) + offset$ $= (6 \times 1024) + 986$ $= 6144 + 986 = 7130$ **Final Answer for (b):** 7130				

4.	Write assembly-level programs (simple mnemonics MOV, ADD, SUB, etc…) to compute $Y = \frac{(A+B)\times C}{D\times E}$ using instruction formats with:
	(a) 3 operands
	(b) 2 operands
	(c) 1 operand
	
	Anskey:
	Assumptions (same for all three):
	• Variables A,B,C,D,E,Y are in memory (word or dword).
	• DIV performs **integer** division: dest = dest / src.
	• No overflow/trap handling shown; introduce wider temporaries if needed.
	; Registers used:
	; r0..r7 are general purpose
	; r0 = (A + B)
	; r1 = C
	; r2 = (D * E)
	; r3 = numerator ((A+B) * C)
	; r4 = Y (result)
	
	; Load inputs
	LD r5, [A] ; r5 = A
	LD r6, [B] ; r6 = B
	LD r1, [C] ; r1 = C
	LD r7, [D] ; r7 = D
	LD r2, [E] ; r2 = E
		10	4	3

```
; (A + B)
ADD  r0, r5, r6      ; r0 = A + B

; Numerator: (A + B) * C
MUL  r3, r0, r1      ; r3 = r0 * r1

; Denominator: D * E
MUL  r2, r7, r2      ; r2 = D * E

; Y = numerator / denominator
DIV  r4, r3, r2      ; r4 = r3 / r2

; Store result
ST   [Y], r4
```

b. 2-operand format (x86-like)

```
; Registers used: R0..R4
; NOTE: two-operand ops overwrite the destination.

MOV  R0, [A]         ; R0 = A
ADD  R0, [B]         ; R0 = A + B

MOV  R1, [C]         ; R1 = C
MUL  R0, R1          ; R0 = (A + B) * C    (R0 ← R0 * R1)

MOV  R2, [D]         ; R2 = D
MUL  R2, [E]         ; R2 = D * E

; Divide numerator by denominator
MOV  R3, R0          ; R3 = numerator
DIV  R3, R2          ; R3 = R3 / R2

MOV  [Y], R3         ; store result
```

3) 1-operand format (Accumulator machine)

Form: operations act on a single implicit accumulator **AC** (and memory).
Typical mnemonics: LOAD X, ADD X, SUB X, MUL X, DIV X, STORE X.
Here MUL X does AC ← AC * M[X], DIV X does AC ← AC / M[X].
; Uses an implicit accumulator AC.
; TEMP1 and TEMP2 are temporary memory locations (for numerator and denominator).

```
; AC = A + B
LOAD  A              ; AC = A
ADD   B              ; AC = A + B

; AC = (A + B) * C
MUL   C              ; AC = (A + B) * C
STORE TEMP1          ; TEMP1 = numerator
```

```
    ; AC = D * E
    LOAD  D             ; AC = D
    MUL   E             ; AC = D * E
    STORE TEMP2         ; TEMP2 = denominator

    ; AC = TEMP1 / TEMP2
    LOAD  TEMP1         ; AC = numerator
    DIV   TEMP2         ; AC = AC / denominator

    STORE Y             ; Y = result
.section .data
A:  .word 0         # fill with your values
B:  .word 0
C:  .word 0
D:  .word 1         # avoid divide-by-zero in examples
E:  .word 1
Y:  .word 0

    .section .t
_start:
    # Load inputs
    la   t0, A       ; t0 = &A
    lw   t1, 0(t0)    ; t1 = A
    la   t0, B
    lw   t2, 0(t0)    ; t2 = B
    la   t0, C
    lw   t3, 0(t0)    ; t3 = C
    la   t0, D
    lw   t4, 0(t0)    ; t4 = D
    la   t0, E
    lw   t5, 0(t0)    ; t5 = E

    # (A + B)
    add  t6, t1, t2          # t6 = A + B

    # Numerator: (A + B) * C
    mul  a0, t6, t3          # a0 = numerator

    # Denominator: D * E
    mul  a1, t4, t5          # a1 = denominator

    # Y = numerator / denominator
    # (Optionally guard against a1==0 before dividing)
    div  a2, a0, a1          # a2 = a0 / a1

    # Store result
    la   t0, Y
    sw   a2, 0(t0)
```

```
    # exit (environment dependent; here just loop)
1:  j   1b
``
_start:
    # R0 = A
    la   t0, A
    lw   s0, 0(t0)          # s0 plays the role of "dst"

    # R0 = R0 + B
    la   t0, B
    lw   t1, 0(t0)
    add  s0, s0, t1         # s0 = A + B

    # R0 = R0 * C
    la   t0, C
    lw   t1, 0(t0)
    mul  s0, s0, t1         # s0 = (A + B) * C   ; numerator in s0

    # R2 = D
    la   t0, D
    lw   s1, 0(t0)

    # R2 = R2 * E
    la   t0, E
    lw   t1, 0(t0)
    mul  s1, s1, t1         # s1 = D * E       ; denominator in s1

    # R3 = R0 / R2
    div  s2, s0, s1         # s2 = numerator / denominator

    la   t0, Y
    sw   s2, 0(t0)

1:  j   1b
``
    .section .bss
    .align 2
TEMP1:  .space 4     # numerator
TEMP2:  .space 4     # denominator

    .section .text
    .globl _start
_start:
    # AC = A
    la   t0, A
    lw   a0, 0(t0)

    # AC = AC + B
    la   t0, B
```

```
    lw   t1, 0(t0)
    add  a0, a0, t1       # (emulates: ADD B)

    # AC = AC * C
    la   t0, C
    lw   t1, 0(t0)
    mul  a0, a0, t1       # (emulates: MUL C)

    # STORE TEMP1  ; numerator
    la   t0, TEMP1
    sw   a0, 0(t0)

    # AC = D
    la   t0, D
    lw   a0, 0(t0)

    # AC = AC * E
    la   t0, E
    lw   t1, 0(t0)
    mul  a0, a0, t1

    # STORE TEMP2  ; denominator
    la   t0, TEMP2
    sw   a0, 0(t0)

    # LOAD TEMP1
    la   t0, TEMP1
    lw   a0, 0(t0)

    # DIV TEMP2   ; AC = AC / M[TEMP2]
    la   t0, TEMP2
    lw   t1, 0(t0)
    div  a0, a0, t1

    # STORE Y
    la   t0, Y
    sw   a0, 0(t0)

  1: j   1b
    beq  a1, x0, .div_by_zero   # if denominator == 0
    div  a2, a0, a1
    j    .after_div
.div_by_zero:
    li   a2, 0              # or some error code
.after_div:
```

5.	Write the sequence of micro-instructions for the following machine instructions: (a) ISZ X, which means the content of location X is incremented by 1; If the result is 0, else the next instruction is skipped.	10	4	3

(b) BSA X, which means the address of the instruction that follows the BSA instruction is saved in location X, and execution continues at that location.
Ans key:

Here are the micro-instruction sequences for a typical basic computer architecture (like the Mano Machine).

(a) ISZ X (Increment and Skip if Zero)

This instruction reads a value from memory, increments it, and writes it back. If the result is zero, the **Program Counter (PC)** is incremented to skip the next instruction.

1. T_4: $DR \leftarrow M[AR]$ (Read the value from memory location X into the Data Register)
2. T_5: $DR \leftarrow DR + 1$ (Increment the Data Register)
3. T_6: $M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$ (Write back to memory; check for zero)

(b) BSA X (Branch and Save Return Address)

This instruction is used for subroutine calls. It saves the return address at location X and then jumps to the instruction at

1. T_4: $M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$ (Store the return address at location X, then increment the Address Register)
2. T_5: $PC \leftarrow AR$, $SC \leftarrow 0$ (Update the Program Counter to $X + 1$ and reset the sequence counter)

Below are concise **micro-instruction sequences** for the two machine instructions, written in the style of Mano's *Basic Computer* (single accumulator) using these registers & signals:

- **PC**: Program Counter
- **AR**: Address Register (effective address of operand)
- **IR**: Instruction Register
- **DR**: Data Register (memory buffer)
- **AC**: Accumulator (not used here)
- **M[AR]**: memory word at address in AR
- **r / w**: memory read / write control

- Clock steps **T0, T1, …** (only the **execute** steps are shown; assume the standard fetch/decode already placed the **effective address in AR**).
  If your notation uses **MDR** instead of **DR**, substitute MDR for DR.

(a) ISZ X

Meaning: Increment the content at location X; **if the new content is 0**, **skip** the next instruction (i.e., PC ← PC + 1).

Execute sequence (with AR = EA):

T3: DR ← M[AR] (r) ; read operand from memory
T4: DR ← DR + 1 ; increment
T5: M[AR] ← DR (w) ; write back the incremented value
T6: if (DR = 0) then PC ← PC + 1 ; skip next instruction on zero result

No flags required beyond testing **DR = 0** (or a Zero flag set by DR ← DR + 1 if your control unit provides it).

b)

T3: M[AR] ← PC (w) ; store return address into X
T4: AR ← AR + 1 ; compute X + 1
T5: PC ← AR ; branch to X + 1

- After T5, the next fetch uses PC = X + 1, so execution proceeds at that location.
- A subroutine placed starting at **X+1** can later return by moving M[X] back into PC.

Cycle	Micro-operations
FETCH T0	`AR ← PC`
FETCH T1	`IR ← M[AR], PC ← PC + 1`
FETCH T2	`Decode, AR ← IR(address)`
INDIRECT T3	`AR ← M[AR]`
ISZ T3	`DR ← M[AR]`
ISZ T4	`DR ← DR + 1`
ISZ T5	`M[AR] ← DR`
ISZ T6	`If DR = 0 then PC ← PC + 1`
BSA T3	`M[AR] ← PC`
BSA T4	`AR ← AR + 1`
BSA T5	`PC ← AR`

****