



**COMSATS University Islamabad**

**Spring 2026**

**EEE341 Introduction to Computer Architecture**

**Midterm**

**Program:** BCS -5

**Total Marks:** 25

**Semester:** Spring 2026

**Friday, 17 April 2026**

**Instructor:** Dr. Muhammad Dilshad Sabir

**Total Time:** 90 min

**1. Answer these short questions briefly and to the point.**

**[2+2+2+2+4]**

1. How the compiler solves the problem of branch range?
2. Why 16-bit offset of branch converted to 32-bit offset?
3. What is the difference between the working of branch and jump instruction?
4. What is the purpose of \$ra register?
5. What is the purpose of offset in load/store and branch instruction? What is range in both cases?

**2. Convert the given C code below in illustration 1 into MIPS assembly language.**

**[13]**

```
main ()
{
  int x, N = 5, acc, PS, NG;
  int A[N] = {12, 8, 6, -74, 99};
  acc = 0;
  PS = 0;
  NG = 0;
  for (x = 0; x < N; x++)
  {
    acc += A[x];

    if (A[x] > 0)
      PS += A[x];
    if (A[x] < 0)
      NG += A[x];
  }
  return 0;
}
```

*Illustration 1: Snip-shot of C program*



COMSATS University Islamabad

Spring 2026

EEE341 Introduction to Computer Architecture

Midterm-Solution

Program: BCS -5

Total Marks: 25

Semester: Spring 2026

Friday, 17 April 2026

Instructor: Dr. Muhammad Dilshad Sabir

Total Time: 90 min

### Question#01

1. **Compiler solves branch range** by replacing a conditional branch with the opposite condition, skipping a `j` (jump) instruction that reaches the far target.
2. **16-bit offset is sign-extended to 32 bits** so the ALU can compute the branch address using  $PC + 4 + (\text{sign\_extended\_offset} \ll 2)$ .
3. **Branch vs jump**: Branch uses PC-relative addressing (conditional, limited range); jump uses absolute addressing (unconditional, larger range via 26-bit immediate or register).
4. *\*\*rapurpose\*\**  
*holdsthereturnaddressforasubroutine,savedby'jal'andusedby'jr'ra` to return.*
5. **Offset purpose & range**
  - a. **Load/store**: offset added to base register → memory address. Range:  $\pm 32,768$  bytes (16-bit signed).
  - b. **Branch**: offset ( $\ll 2$ ) added to  $PC+4$  → target address. Range:  $\pm 128$  Kbytes (16-bit signed, word-aligned).

### Question#02

```
data
A:      .word    12, 8, 6, -74, 99      # array A[5]
N:      .word    5                      # constant N

.text
.globl main

main:
# Prologue: reserve stack space (optional, but good practice)
addi    $sp, $sp, -4
sw      $ra, 0($sp)

# Initialize variables
li      $t0, 0      # x = 0
li      $t1, 0      # acc = 0
```

```

li      $t2, 0          # PS = 0
li      $t3, 0          # NG = 0
lw      $t4, N          # $t4 = N = 5
la      $t5, A          # $t5 = base address of A

loop:
    # Loop condition: x < N
    bge   $t0, $t4, end_loop

    # Compute acc += A[x]
    sll   $t6, $t0, 2     # $t6 = x * 4
    add   $t6, $t5, $t6    # $t6 = &A[x]
    lw    $t7, 0($t6)     # $t7 = A[x]
    add   $t1, $t1, $t7    # acc += A[x]

    # if (A[x] > 0) PS += A[x]
    blez  $t7, check_neg  # if A[x] <= 0 skip
    add   $t2, $t2, $t7    # PS += A[x]
    j     check_pos       # skip NG part

check_neg:
    # if (A[x] < 0) NG += A[x]
    bgez  $t7, next_iter   # if A[x] >= 0 skip
    add   $t3, $t3, $t7    # NG += A[x]

check_pos:
next_iter:
    addi  $t0, $t0, 1     # x++
    j     loop

end_loop:
    # return 0
    li    $v0, 0

    # Epilogue
    lw    $ra, 0($sp)
    addi  $sp, $sp, 4
    jr    $ra

```