Ex1:

Code:

A screen shot of a computer

Description automatically generated

Data segment:

A screenshot of a computer

Description automatically generated

Ex 1.1, 1.2:

Code:

A screenshot of a computer code

Description automatically generated

Solution:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Ex 2:

Code:

A screenshot of a computer program

Description automatically generated

Solution:

A screenshot of a computer

Description automatically generated

Ex 3:

Code:

A screenshot of a computer

Description automatically generated

Solution:

//slli x30, x5, 2

int offset\_A = f \* 4;

//add x30, x10, x30

int\* address\_A\_f = A + f;

//slli x31, x6, 2

int\* address\_A\_f = A + f;

//add x31, x11, x31

int\* address\_B\_g = B + g;

//lw x5, 0(x30)

f = A[f];

//addi x12, x30, 8

int\* address\_A\_f2 = address\_A\_f + 2;

//lw x30, 0(x12)

int temp\_A\_f2 = A[f + 2];

//add x30, x30, x5

int sum\_A\_f\_A\_f2 = A[f + 2] + f;

//lw x30, 0(x31)

int value\_B\_g = B[g];

Ex 7:

sw x5, 32(x30)

* This is a Store Word (SW) instruction, S-type format (Store-type). It stores the value from register x5 into memory at the address x30 + 32.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 - 25 | 24 - 20 | 19 - 15 | 14 - 12 | 11 - 7 | 6 - 0 |
| imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode |

Where:

* imm[11:5]: The upper 7 bits of the immediate (offset)
* rs2: The source register to store (x5 in this case)
* rs1: The base register (x30 in this case)
* funct3: This is a 3-bit field that defines the type of operation, for sw it is 010.
* imm[4:0]: The lower 5 bits of the immediate (offset)
* opcode: The opcode for sw, which is 0100011.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| imm[11:5] | rs2 (x5) | rs1 (x30) | funct3 | imm[4:0] | opcode |
| 0000000 | 00101 | 11110 | 010 | 10000 | 0100011 |

* In binary: 0000000 00101 11110 010 10000 0100011
* In hexadecimal: 0x005F1023

Ex :

Code:

.data

    msg: .asciiz "Enter an integer: "

    odd: .asciiz "odd"

    even: .asciiz "even"

.text

main:

print\_msg:

    li a7, 4              # syscall for print string

    la a0, msg         # load address of the prompt

    ecall

input\_integer:

    li a7, 5              # syscall for reading integer

    ecall

    addi t0, a0, 0            # move the input to t0

check:

    li      t1, 2              # load 2 into t1

    rem     t2, t0, t1         # compute t0 % 2, result in t2

    beq     t2, x0, is\_even    # if t2 == 0, branch to is\_even

is\_odd:

    li      a7, 4              # syscall for print string

    la      a0, odd        # load address of "odd" message

    ecall

    j       exit               # jump to exit

is\_even:

    li      a7, 4              # syscall for print string

    la      a0, even       # load address of "even" message

    ecall

exit:

    li      a7, 10             # syscall for exit

    ecall

Ex :

Code:

.text

main:

    li a7, 5 # read an int to a0

    ecall

    beq a0, zero, exit # exit if input = 0

    jal abs # call abs procedure

    li a7, 1 # print return value

    ecall

    j main

exit:

    li a7, 10 # terminate, WHY is this necessary?

    ecall # see system service 10

end\_main:

abs:

    bge a0, zero, done # if a0>=0 done

    sub a0, zero, a0 # else negate a0

done:

    jr ra

Result:

A screenshot of a computer

Description automatically generated

Ex :

Code:

.data

    msg:     .asciiz "Enter a positive integer: "

    result: .asciiz "The sum is: "

.text

main:

    # Print prompt message to ask for input

    la      a0, msg         # load address of prompt message

    li      a7, 4              # syscall for print string (a7 = 4)

    ecall                      # make the system call to print string

    # Read integer input from the console

    li      a7, 5              # syscall for reading integer (a7 = 5)

    ecall                      # make the system call to read integer

    # Move the input value from a0 to a7 for the procedure

    mv      a7, a0             # move the input (n) to a7

    # Call the sum procedure

    jal     ra, calculate\_sum       # jump and link to calc\_sum procedure

    # Print the result message

    la      a0, result         # load address of result message

    li      a7, 4              # syscall for print string (a7 = 4)

    ecall                      # make system call to print string

    # Print the sum result (in a0)

    mv      a0, a0             # move result (sum) to a0 (already in a0)

    li      a7, 1              # syscall for print integer (a7 = 1)

    ecall                      # make system call to print integer

    # Exit the program

    li      a7, 10             # syscall for exit (a7 = 10)

    ecall                      # make system call to exit

# Procedure to calculate the sum of numbers from 1 to n

calculate\_sum:

    li      a0, 0              # initialize sum to 0

    li      t0, 1              # initialize counter i to 1

sum\_loop:

    bgt     t0, a7, end\_sum    # if i > n, exit loop

    add     a0, a0, t0         # sum = sum + i

    addi    t0, t0, 1          # i = i + 1

    j       sum\_loop           # repeat the loop

end\_sum:

    jr      ra                 # return to the caller

Ex:

Code:

.data

    str:    .space 255

    mes1:   .asciiz "Input the string from keyboard"

    mes2:   .asciiz "The length of the string is: "

.text

main:

get\_string:

    li a7, 54          #service number to InputDialogString

    la a0, mes1        #load address of message on dialog to a0

    la a1, str         #load address of input string to a1

    li a2, 255          #load buffer to a2 (50)

    ecall              #system call

get\_length:

    la a0, str         #a0 = address of str[0]

    li t0, 0           #t0 = i = 0

check\_char:

    add t1, a0, t0     #t1 = a0 + t0   //address of str[0 + i]

    lb t2, 0(t1)       #t2 = str[i]

    beq t2, zero, end\_of\_str    #check end of str

    addi t0, t0, 1     #t0 = t0 + 1    //i = i + 1

    j check\_char

end\_of\_str:

end\_of\_get\_length:

    addi t0, t0, -1    #avoid counting new line character (\n)

print\_length:

    li a7, 56          #service number to MessageDialogInt

    la a0, mes2        #load address of message on dialog to a0

    add a1, t0, zero   #load a1 = length of string

    ecall              #system call

Result:

Input:

A screenshot of a computer

Description automatically generated

Output:

A screenshot of a computer

Description automatically generated