**RISC PROJECT**

**REPORT**

Topic: Depiction of Instruction Fetch and Decoding phase of RISC V processor

By,

C.P.Sindhu(PES1UG21EC071)

Dhanush M.R.(PES1UG21EC084)

**Aim:** Depicting instruction fetch and decode stage of a RISC V processor using assembly level program

**CODE:**  
.data # This section is for data declaration

imem: .word 0x00000613 #0x3e112423 #0x003100b3,0x403100b3,0x3e810093,0x3e812083 # 'imem' is an array of words initialized with the given values

.text # This section is for code

li x15,1 # Load immediate value 1 into register x15

la x11,imem # Load the address of 'imem' into register x11

li x12,0 # Load immediate value 0 into register x12

statement: .string "Decoded instruction: " # Declare a string named 'statement' with the given value

la a0,statement # Load the address of 'statement' into register a0 (argument register 0)

li a7,4 # Load immediate value 4 into register a7 (system call number for print string)

ecall # Perform system call

#extracting opcode

lw x10,0(x11) # Load word from memory at address in x11 into x10

li x1, 0b00000000000000000000000001111111 # Load immediate value (mask for opcode) into x1

and x1, x10, x1 # Bitwise AND operation between x10 and x1, result stored in x1

sw x1,0(x12) # Store word in x1 into memory at address in x12

li x18,51 #opcode for R type

li x19,19 #opcode for immediate type

li x20,3 #opcode for load type

li x21,35 #opcode for store type

li x22,103 # Load immediate opcode values into registers x18 to x22

beq x1,x18,r\_type

beq x1,x19,i\_type

beq x1,x20,i\_type

beq x1,x22,i\_type

beq x1,x21,s\_type # Compare opcode in x1 with known opcode values, branch to corresponding label if equal

r\_type: # Label for R-type instructions

# Extract the rd

li x31, 0b00000000000000000000111110000000 # Load immediate value (mask for rd) into x31

and x31, x10, x31 # Bitwise AND operation between x10 and x31, result stored in x31

srli x31, x31, 7 # Shift right logical immediate, shift x31 right by 7 bits, result stored in x31

sw x31, 1(x12) # Store word in x31 into memory at address in x12 offset by 1

# Extract the function code (funct3)

li x30, 0b00000000000000000111000000000000 # Load immediate value (mask for funct3) into x30

and x30, x10, x30 # Bitwise AND operation between x10 and x30, result stored in x30

srli x30, x30, 12 # Shift right logical immediate, shift x30 right by 12 bits, result stored in x30

sw x30, 2(x12) # Store word in x30 into memory at address in x12 offset by 2

# Extract the source register 1 (rs1)

li x3, 0b00000000000011111000000000000000 # Load immediate value (mask for rs1) into x3

and x3, x10, x3 # Bitwise AND operation between x10 and x3, result stored in x3

srli x3, x3, 15 # Shift right logical immediate, shift x3 right by 15 bits, result stored in x3

sw x3, 3(x12) # Store word in x3 into memory at address in x12 offset by 3

# Extract the source register 2 (rs2)

li x3, 0b00000001111100000000000000000000 # Load immediate value (mask for rs2) into x3

and x3, x10, x3 # Bitwise AND operation between x10 and x3, result stored in x3

srli x3, x3, 20 # Shift right logical immediate, shift x3 right by 20 bits, result stored in x3

sw x3, 4(x12) # Store word in x3 into memory at address in x12 offset by 4

# Extract the func 7

li x5, 0b11111110000000000000000000000000 # Load immediate value (mask for func7) into x5

and x5, x10, x5 # Bitwise AND operation between x10 and x5, result stored in x5

srli x5,x5,25 # Shift right logical immediate, shift x5 right by 25 bits, result stored in x5

sw x5, 5(x12) # Store word in x5 into memory at address in x12 offset by 5

li s0, 1

li t1, 4

li t2, 5

li t3, 6

li t4, 7 # Load immediate values into registers s0 to t4

beq x30,x0,add\_sub

beq x30,s0,shift\_ll

beq x30,t1,x\_or

beq x30,t2,sr\_l\_a

beq x30,t3,o\_r

beq x30,t4,a\_nd # Compare funct3 in x30 with known funct3 values, branch to corresponding label if equal

add\_sub:

li s0,32

sw s0,0(sp)

lw s0,0(sp)

beq x5,x0,a\_dd

beq x5,s0,s\_ub

a\_dd:

instr1: .string "add"

la a0,instr1

li a7,4

ecall

j branch\_done

s\_ub:

instr2: .string "sub"

la a0,instr2

li a7,4

ecall

j branch\_done

branch\_done: j r\_done # Label for end of add/sub block, jump to r\_done

shift\_ll:

instr3: .string "sll"

la a0,instr3

li a7,4

ecall

j r\_done # Jump to r\_done after executing shift left logical

x\_or:

instr4: .string "xor"

la a0,instr4

li a7,4

ecall

j r\_done # Jump to r\_done after executing xor

sr\_l\_a:

li s0,32

sw s0,0(sp)

lw s0,0(sp)

beq x5,x0,sr\_l

beq x5,s0,sr\_a

sr\_l:

instr5: .string "srl"

la a0,instr5

li a7,4

ecall

j if\_done

sr\_a:

instr6: .string "sra"

la a0,instr6

li a7,4

ecall

j if\_done

if\_done: j r\_done # Label for end of sr\_l\_a block, jump to r\_done

o\_r:

instr7: .string " or"

la a0,instr7

li a7,4

ecall

j r\_done # Jump to r\_done after executing or

a\_nd:

instr8: .string "and"

la a0,instr8

li a7,4

ecall

j r\_done # Jump to r\_done after executing and

r\_done: j print # Label for end of r\_type block, jump to print

s\_type: # Label for S-type instructions

# Extract the imm1 (imm1)

li x31, 0b00000000000000000000111110000000 # Load immediate value (mask for imm1) into x31

and x31, x10, x31 # Bitwise AND operation between x10 and x31, result stored in x31

srli x31, x31, 7 # Shift right logical immediate, shift x31 right by 7 bits, result stored in x31

# Extract the function code (funct3)

li x4, 0b00000000000000000111000000000000 # Load immediate value (mask for funct3) into x4

and x4, x10, x4 # Bitwise AND operation between x10 and x4, result stored in x4

srli x4, x4, 12 # Shift right logical immediate, shift x4 right by 12 bits, result stored in x4

sw x4, 1(x12) # Store word in x4 into memory at address in x12 offset by 1

# Extract the source register 1 (rs1)

li x3, 0b00000000000011111000000000000000 # Load immediate value (mask for rs1) into x3

and x3, x10, x3 # Bitwise AND operation between x10 and x3, result stored in x3

srli x3, x3, 15 # Shift right logical immediate, shift x3 right by 15 bits, result stored in x3

sw x3, 2(x12) # Store word in x3 into memory at address in x12 offset by 2

# Extract the source register 2 (rs2)

li x3, 0b00000001111100000000000000000000 # Load immediate value (mask for rs2) into x3

and x3, x10, x3 # Bitwise AND operation between x10 and x3, result stored in x3

srli x3, x3, 20 # Shift right logical immediate, shift x3 right by 20 bits, result stored in x3

sw x3, 3(x12) # Store word in x3 into memory at address in x12 offset by 3

# Extract the immediate value 2 (imm2)

li x30, 0b11111110000000000000000000000000 # Load immediate value (mask for imm2) into x30

and x30, x10, x30 # Bitwise AND operation between x10 and x30, result stored in x30

srli x30,x30,20 # Shift right logical immediate, shift x30 right by 20 bits, result stored in x30

add x30,x30,x31 # Add x30 and x31, result stored in x30

sw x30, 4(x12) # Store word in x30 into memory at address in x12 offset by 4

li s0, 1 # Load immediate value 1 into register s0

li t1, 2 # Load immediate value 2 into register t1

beq x4,x0,s\_b # Compare funct3 in x4 with 0, branch to s\_b if equal

beq x4,s0,s\_h # Compare funct3 in x4 with s0, branch to s\_h if equal

beq x4,t1,s\_w # Compare funct3 in x4 with t1, branch to s\_w if equal

s\_b:

instr9: .string "sb " # Declare a string named 'instr9' with the given value

la a0,instr9 # Load the address of 'instr9' into register a0 (argument register 0)

li a7,4 # Load immediate value 4 into register a7 (system call number for print string)

ecall # Perform system call

j s\_done # Jump to s\_done

s\_h:

instr10: .string "sh " # Declare a string named 'instr10' with the given value

la a0,instr10 # Load the address of 'instr10' into register a0 (argument register 0)

li a7,4 # Load immediate value 4 into register a7 (system call number for print string)

ecall # Perform system call

j s\_done # Jump to s\_done

s\_w:

instr11: .string "sw " # Declare a string named 'instr11' with the given value

la a0,instr11 # Load the address of 'instr11' into register a0 (argument register 0)

li a7,4 # Load immediate value 4 into register a7 (system call number for print string)

ecall # Perform system call

j s\_done # Jump to s\_done

s\_done:j print # Label for end of s\_type block, jump to print

i\_type: # Label for I-type instructions

# Extract the destination register (rd)

li x31, 0b00000000000000000000111110000000 # Load immediate value (mask for rd) into x31

and x31, x10, x31 # Bitwise AND operation between x10 and x31, result stored in x31

srli x31, x31, 7 # Shift right logical immediate, shift x31 right by 7 bits, result stored in x31

sw x31, 1(x12) # Store word in x31 into memory at address in x12 offset by 1

# Extract the source register (rs1)

li x30, 0b00000000000011111000000000000000 # Load immediate value (mask for rs1) into x30

and x30, x10, x30 # Bitwise AND operation between x10 and x30, result stored in x30

srli x30, x30, 15 # Shift right logical immediate, shift x30 right by 15 bits, result stored in x30

sw x30, 2(x12) # Store word in x30 into memory at address in x12 offset by 2

# Extract the function code (funct3)

li x27, 0b00000000000000000111000000000000 # Load immediate value (mask for funct3) into x27

and x27, x10, x27 # Bitwise AND operation between x10 and x27, result stored in x27

srli x27, x27, 12 # Shift right logical immediate, shift x27 right by 12 bits, result stored in x27

sw x27, 3(x12) # Store word in x27 into memory at address in x12 offset by 3

# Extract the immediate value (imm)

li x26, 0b11111111111100000000000000000000 # Load immediate value (mask for imm) into x26

and x26, x10, x26 # Bitwise AND operation between x10 and x26, result stored in x26

srli x26,x26,20 # Shift right logical immediate, shift x26 right by 20 bits, result stored in x26

sw x26, 4(x12) # Store word in x26 into memory at address in x12 offset by 4

li s0, 3 # Load immediate value 3 into register s0

li t1, 19 # Load immediate value 19 into register t1

li t2, 103 # Load immediate value 103 into register t2

beq x1,s0,load # Compare opcode in x1 with s0, branch to load if equal

beq x1,t1,imm # Compare opcode in x1 with t1, branch to imm if equal

beq x1,t2,jump # Compare opcode in x1 with t2, branch to jump if equal

load:

li s0, 1 # Load immediate value 1 into register s0

li t1, 2 # Load immediate value 2 into register t1

li t2, 4 # Load immediate value 4 into register t2

li t3, 5 # Load immediate value 5 into register t3

beq x27,x0,l\_b # Compare funct3 in x27 with 0, branch to l\_b if equal

beq x27,s0,l\_h # Compare funct3 in x27 with s0, branch to l\_h if equal

beq x27,t1,l\_w # Compare funct3 in x27 with t1, branch to l\_w if equal

beq x27,t2,l\_bu # Compare funct3 in x27 with t2, branch to l\_bu if equal

beq x27,t3,l\_hu # Compare funct3 in x27 with t3, branch to l\_hu if equal

l\_b:

instr12: .string "lb " # Declare a string named 'instr12' with the given value

la a0,instr12 # Load the address of 'instr12' into register a0 (argument register 0)

li a7,4 # Load immediate value 4 into register a7 (system call number for print string)

ecall # Perform system call

j i\_done # Jump to i\_done

l\_h:

instr13: .string "lh " # Declare a string named 'instr13' with the given value

la a0,instr13 # Load the address of 'instr13' into register a0 (argument register 0)

li a7,4 # Load immediate value 4 into register a7 (system call number for print string)

ecall # Perform system call

j i\_done # Jump to i\_done

l\_w:

instr14: .string "lw " # Declare a string named 'instr14' with the given value

la a0,instr14 # Load the address of 'instr14' into register a0 (argument register 0)

li a7,4 # Load immediate value 4 into register a7 (system call number for print string)

ecall # Perform system call

j i\_done # Jump to i\_done

l\_bu:

instr15: .string "lbu" # Declare a string named 'instr15' with the given value

la a0,instr15 # Load the address of 'instr15' into register a0 (argument register 0)

li a7,4 # Load immediate value 4 into register a7 (system call number for print string)

ecall # Perform system call

j i\_done # Jump to i\_done

l\_hu:

instr16: .string "lhu" # Declare a string named 'instr16' with the given value

la a0,instr16 # Load the address of 'instr16' into register a0 (argument register 0)

li a7,4 # Load immediate value 4 into register a7 (system call number for print string)

ecall # Perform system call

j i\_done # Jump to i\_done

imm:

li s0, 1 # Load immediate value 1 into register s0

li t1, 4 # Load immediate value 4 into register t1

li t2, 6 # Load immediate value 6 into register t2

li t3, 7 # Load immediate value 7 into register t3

beq x4,x0,add\_i # Compare funct3 in x4 with 0, branch to add\_i if equal

beq x4,s0,sll\_i # Compare funct3 in x4 with s0, branch to sll\_i if equal

beq x4,t1,xor\_i # Compare funct3 in x4 with t1, branch to xor\_i if equal

beq x4,t3,and\_i # Compare funct3 in x4 with t3, branch to and\_i if equal

add\_i:

instr17: .string "addi " # Declare a string named 'instr17' with the given value

la a0,instr17 # Load the address of 'instr17' into register a0 (argument register 0)

li a7,4 # Load immediate value 4 into register a7 (system call number for print string)

ecall # Perform system call

j i\_done # Jump to i\_done

sll\_i:

instr18: .string "slli " # Declare a string named 'instr18' with the given value

la a0,instr18 # Load the address of 'instr18' into register a0 (argument register 0)

li a7,4 # Load immediate value 4 into register a7 (system call number for print string)

ecall # Perform system call

j i\_done # Jump to i\_done

xor\_i:

instr19: .string "xori " # Declare a string named 'instr19' with the given value

la a0,instr19 # Load the address of 'instr19' into register a0 (argument register 0)

li a7,4 # Load immediate value 4 into register a7 (system call number for print string)

ecall # Perform system call

j i\_done # Jump to i\_done

and\_i:

instr21: .string "andi " # Declare a string named 'instr21' with the given value

la a0,instr21 # Load the address of 'instr21' into register a0 (argument register 0)

li a7,4 # Load immediate value 4 into register a7 (system call number for print string)

ecall # Perform system call

j i\_done # Jump to i\_done

jump:

instr22: .string "jalr " # Declare a string named 'instr22'

la a0,instr22

li a7,4

ecall

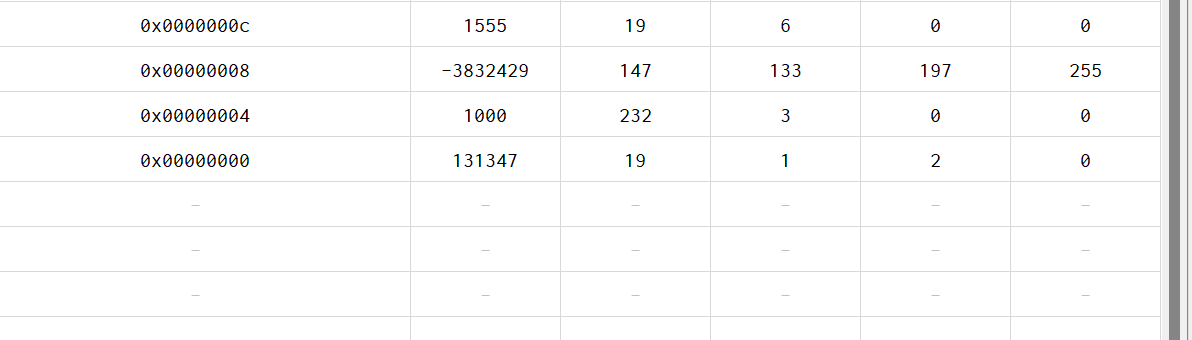
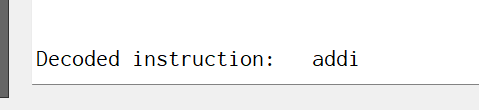
j i\_done

i\_done:j print

print: nop

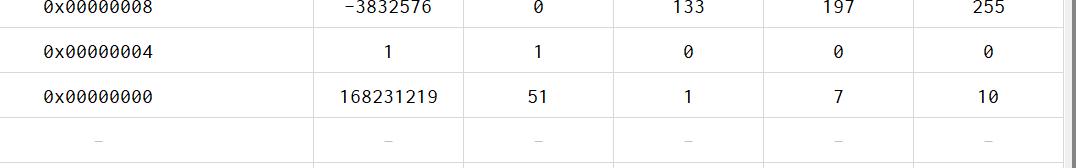
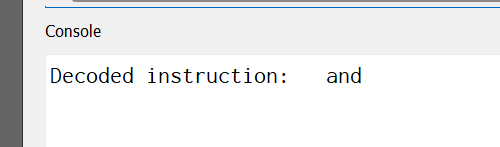
**Results:**

1. Machine code: 0x3e810093



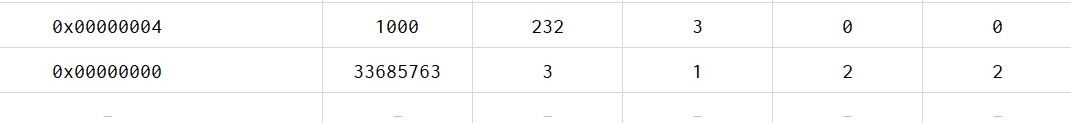
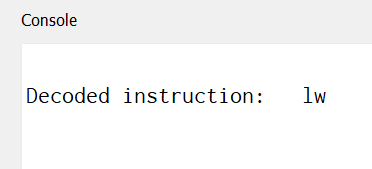
* addi x1,x2,1000

1. Machine Code: 0x001570b3



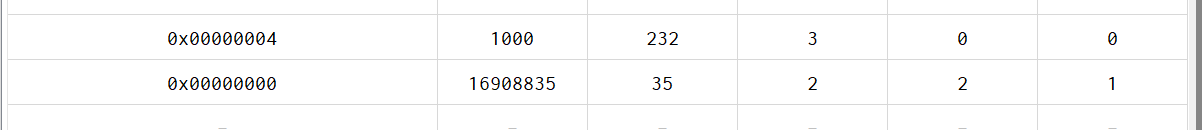
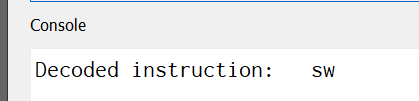
* and x1,x1,x10

1. Machine Code: 0x3e812083



* lw x1,1000(x2)

1. Machine Code : 0x3e112423



* sw x1,1000(x2)

Successfully depicted instruction fetch and instruction decode.

We spent an overall of a weeks time in implementing the above.