ISA & Datapath: Counting Occurrences of a Value in an Unsorted Array

By Venkat Sai Sesha Reddy Dugasani

Shiva Kumar Reddy Rangapuram

Table of Contents

AIM

- To design an instruction set architecture and pipeline implementation of the architecture.

- Input: An unsorted array A = [a1, a2, ….an], size of the array n, a value ‘v’
- Output: How many items in the array are equal to v?

Assumptions

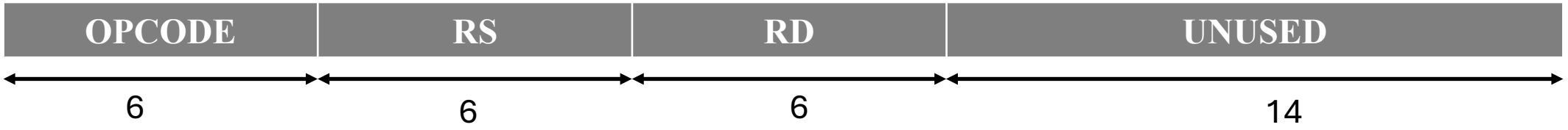
- Word length is exactly four bytes.
- 64 registers are available in the processor, each 32 bits (2's complement).
- Memory addresses are 32 bits.
- Memory access and arithmetic operations take 2 ns each.
- Unused bits in the instruction format are assumed to be set to zero.
- Array access is performed using a register-based approach, where the array index is stored in a dedicated register. This approach is compatible with pipelining and facilitates efficient memory access scheduling and pipeline management.
- All initial values required by the program are directly loaded from predetermined memory addresses into registers at the start of program execution.
- The result of the computation is stored directly into a dedicated register (e.g., R4) instead of using a separate store instruction to write it back to memory.

Assumptions(Continued)

- The status register, containing flags set after a comparison operation, is integrated into the ALU.
- The Control Unit works with the CMP instruction and Branch instruction directly based on the status register value.

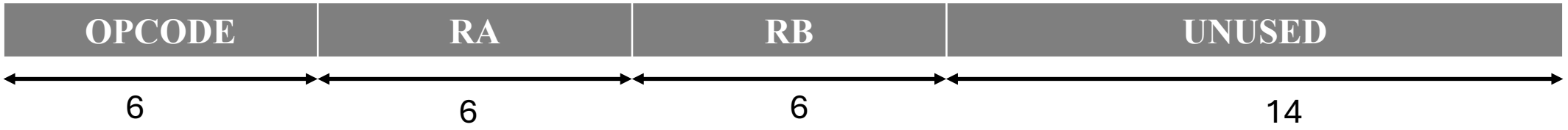
Instruction Set Architecture(ISA)

- Load Rd, Rs



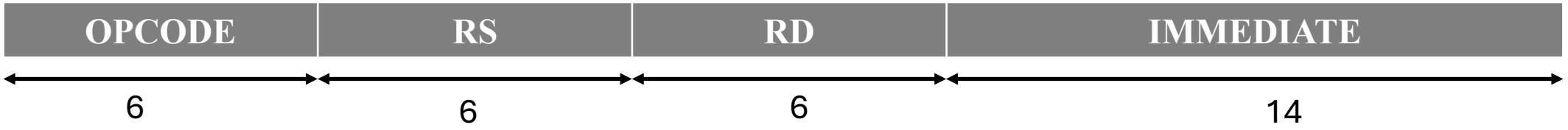
Instruction Set Architecture(ISA)

- Compare Ra, Rb



Instruction Set Architecture(ISA)

- ADDI Rd, Rs, Immediate



Instruction Set Architecture(ISA)

- Jump Offset

Instruction Set Architecture(ISA)

- Branch Offset

Difference between Jump offset, and Branch offset ?

Assembly Code

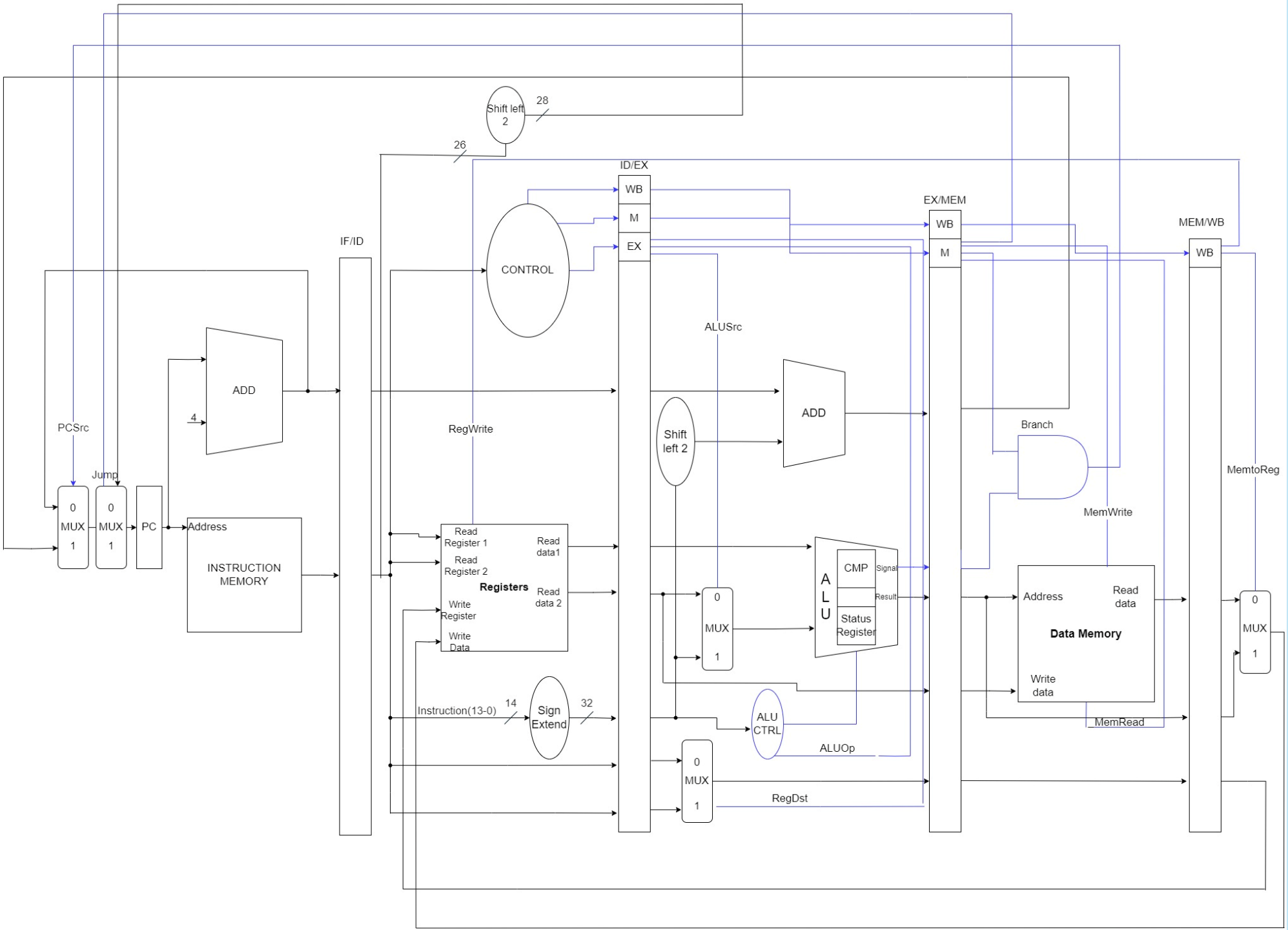
- LOAD R1, A ; Load the base address of the array into R1
- LOAD R2, n ; Load the size of the array into R2
- LOAD R3, v ; Load the value v into R3
- LOAD R4, #0 ; Initialize the count of matching elements to 0
- LOAD R5, #0 ; Initialize an index i to 0 (to iterate over the array)

LOOP:
- CMP R5, R2 ; Compare i with n and set status flags
- BRANCH END ; If previous comparison was equal, jump to END
- LOAD R6, [R1+R5] ; Load A[i] into R6
- CMP R6, R3 ; Compare A[i] with v and set status flags
- BRANCH MATCH ; If previous comparison was equal, jump to MATCH
- ADDI R5, R5, #1 ; Increment i using ADDI
- JMP LOOP ; Jump back to the start of the loop

MATCH:
- ADDI R4, R4, #1 ; Increment the count because A[i] == v
- ADDI R5, R5, #1 ; Increment i using ADDI (continue to the next iteration)
- JMP LOOP ; Jump back to the start of the loop

END:
- ; Result is now in register R4
- ; Continue with other operations using the result in R4

Datapath and Control Signals...


Shift left 2
28
26
ID/EX
WB
M
EX
EX/MEM
WB
M
MEM/WB
WB
IF/ID
CONTROL
ALUSrc
ADD
4
PCSrc
Jump
RegWrite
Shift left 2
ADD
Branch
MemtoReg
MemWrite
0
MUX
1
0
MUX
1
PC
Address
INSTRUCTION MEMORY
Read Register 1
Read Register 2
Registers
Write Register
Write Data
Read data1
Read data 2
0
MUX
1
ALU
CMP
Signal
Result
Status Register
Address
Read data
Data Memory
Write data
MemRead
Instruction(13-0)
14
Sign Extend
32
ALU CTRL
ALUOp
0
MUX
1
RegDst
0
MUX
1

Performance Analysis

Instruction Count:-

- Initialization: 5 instructions for loading values.
- Loop Execution: 5 instructions per iteration.
- Total Instruction Count: (5 + 5n), assuming n loop iterations.

Total Execution Time=Instruction Count × CPI × Cycle Time

Total Execution Time=(5+5n) × 1 × 1ns

Conclusion:

- Total Execution Time: (5 + 5n) nanoseconds.
- CMP and BEQ operations are combined as they work together.
- This estimation assumes ideal conditions and no memory latency issues.

THANK YOU