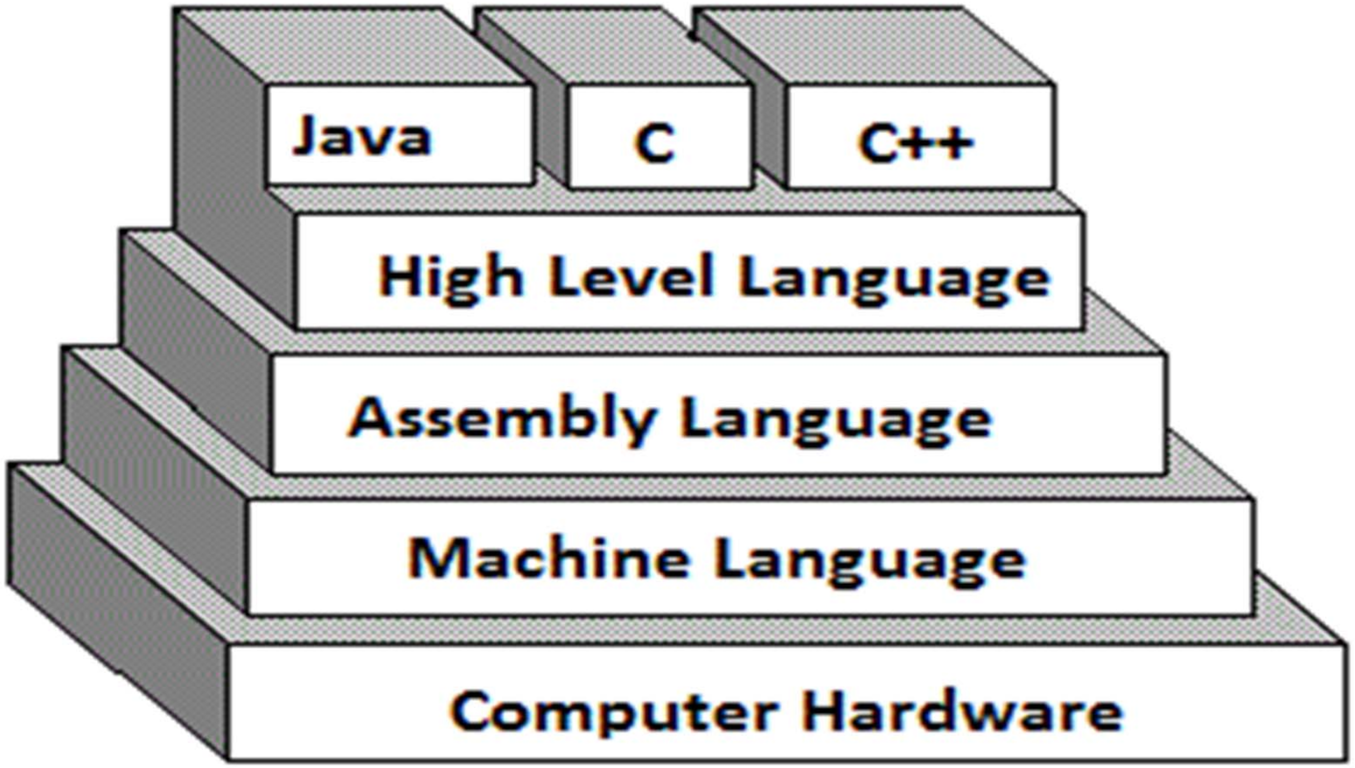
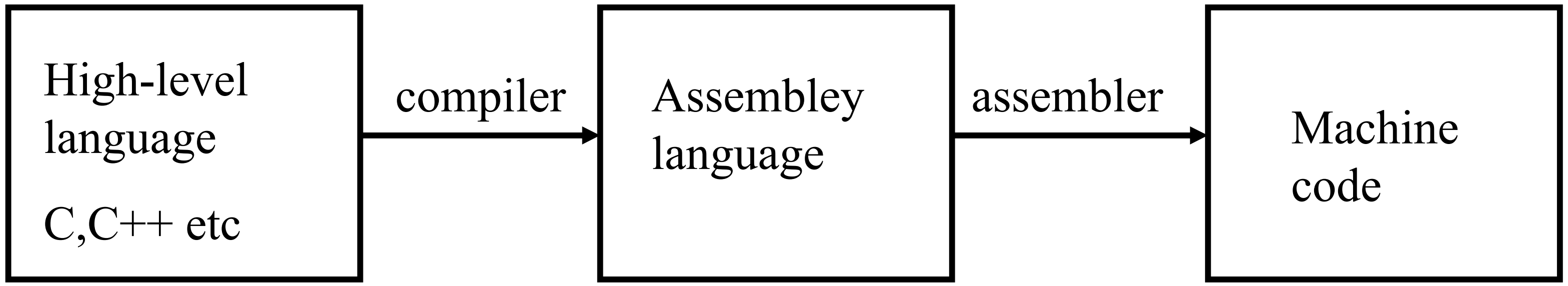
Computer Architectures

- Levels of Programming Languages
- Instruction Set Architecture (ISA)
- Central Processing Unit (CPU)
- The Datapath

Levels of Programming Languages



Computer Programming



```
#include <stdio.h>
 int main()
{
 int a, b, c;
 a = 5; b = 7;
 c = a + b;
printf("%d + %d = %d\n", a, b, c);
return 0;
}
```

```
MOV A, 5
MOV B, 7
ADD A, B
STOP
```

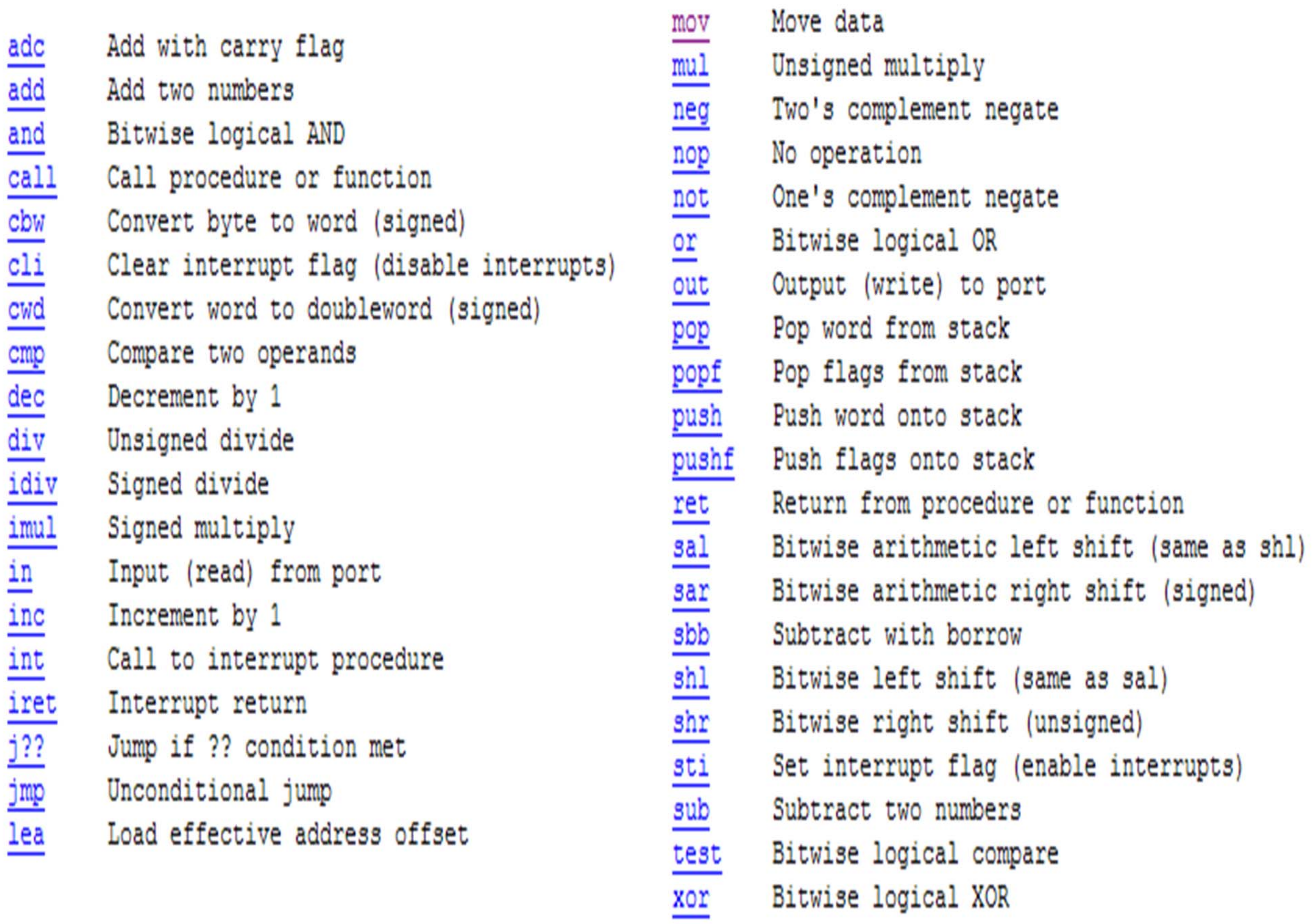
```
1010
1100
1000
1111
```

Programmers View

At the lowest level, a programming language consists of a collection of instructions consisting of 1's and 0's.

This is more easily handled by programmers using assembly language. Each machine code instruction has an intuitive name such as ADD, MOVE, JUMP.

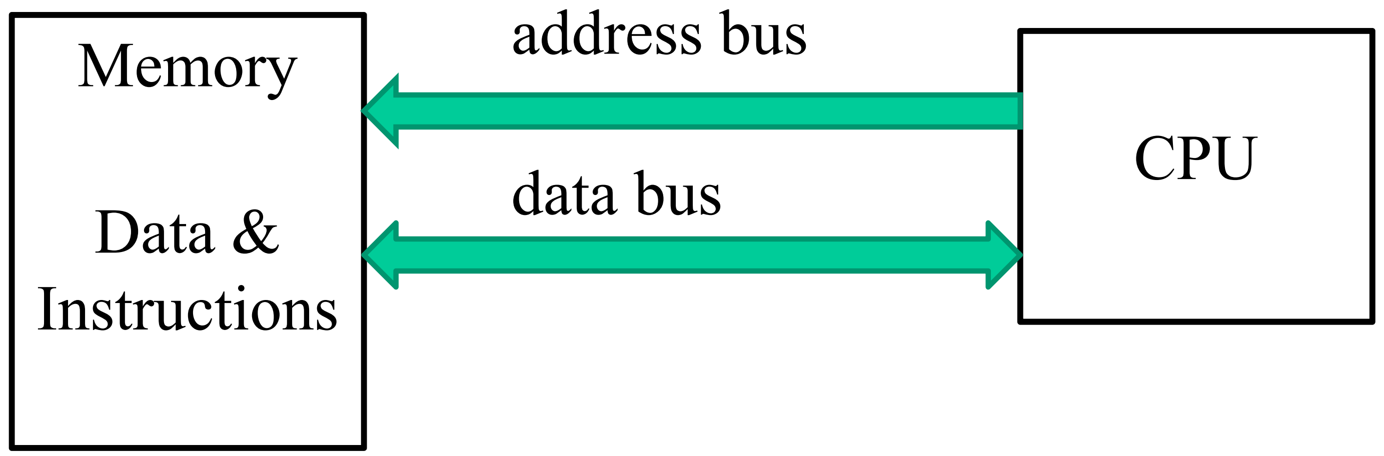
The collection of instructions that a machine supports and their relationship to the registers and functional blocks of the hardware is known as the Instruction Set Architecture (ISA) of the machine.



adc	Add with carry flag
add	Add two numbers
and	Bitwise logical AND
call	Call procedure or function
cbw	Convert byte to word (signed)
cli	Clear interrupt flag (disable interrupts)
cwd	Convert word to doubleword (signed)
cmp	Compare two operands
dec	Decrement by 1
div	Unsigned divide
idiv	Signed divide
imul	Signed multiply
in	Input (read) from port
inc	Increment by 1
int	Call to interrupt procedure
iret	Interrupt return
j??	Jump if ?? condition met
jmp	Unconditional jump
lea	Load effective address offset
mov	Move data
mul	Unsigned multiply
neg	Two's complement negate
nop	No operation
not	One's complement negate
or	Bitwise logical OR
out	Output (write) to port
pop	Pop word from stack
popf	Pop flags from stack
push	Push word onto stack
pushf	Push flags onto stack
ret	Return from procedure or function
sal	Bitwise arithmetic left shift (same as shl)
sar	Bitwise arithmetic right shift (signed)
sbb	Subtract with borrow
shl	Bitwise left shift (same as sal)
shr	Bitwise right shift (unsigned)
sti	Set interrupt flag (enable interrupts)
sub	Subtract two numbers
test	Bitwise logical compare
xor	Bitwise logical XOR

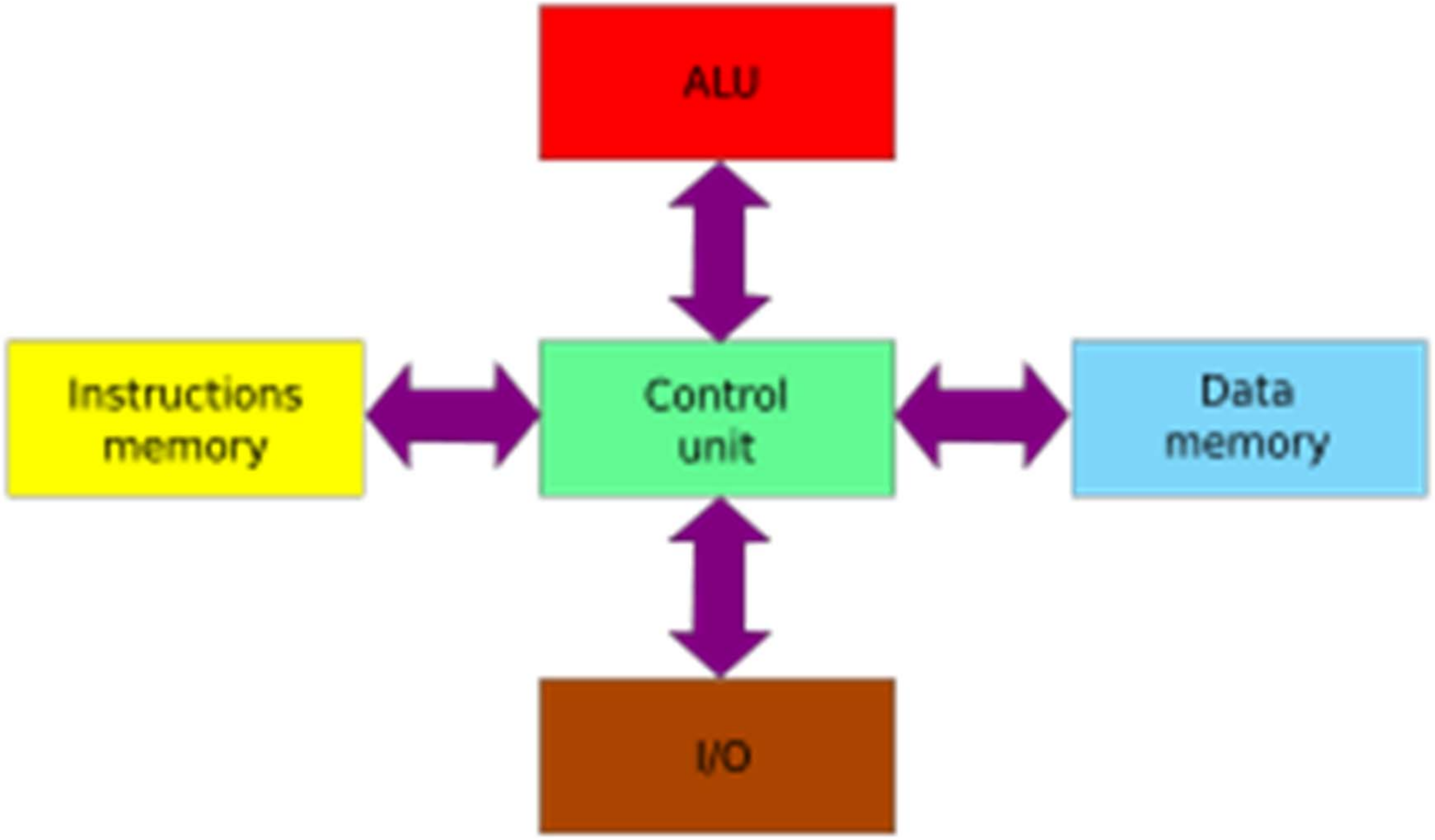
Von Neumann Architecture

Instructions and data both consist of strings of binary digits and can be stored in the same format. In the Von Neumann model, the program and data are stored in a single memory and share a common bus. The programmer must ensure the correct interpretation.



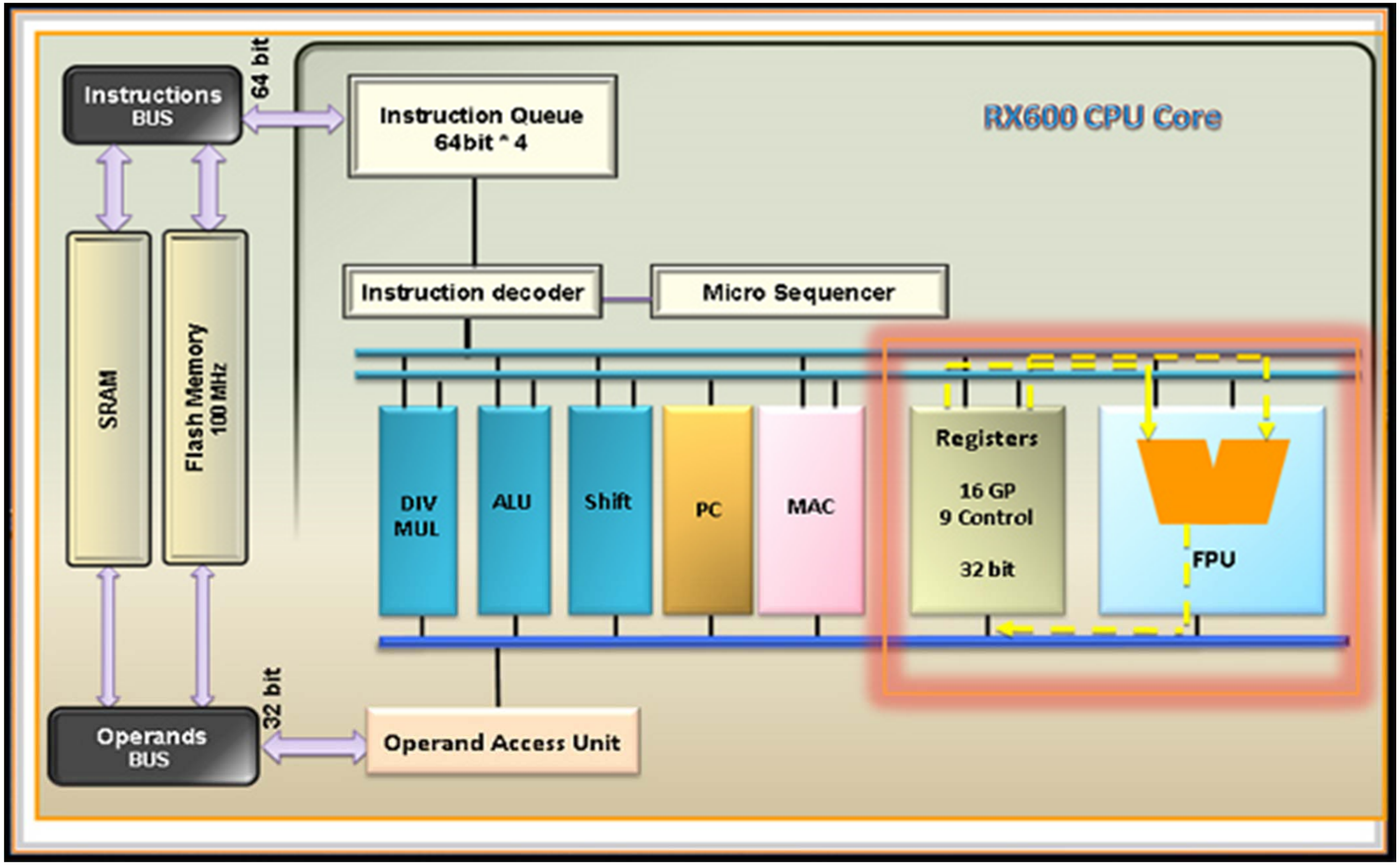
Compiled programs are copied from the hard disk to memory. The CPU reads instructions and data from the memory, executes the instructions and writes data back to memory.

Harvard Architecture



Separate interface for instruction memory and data memory.

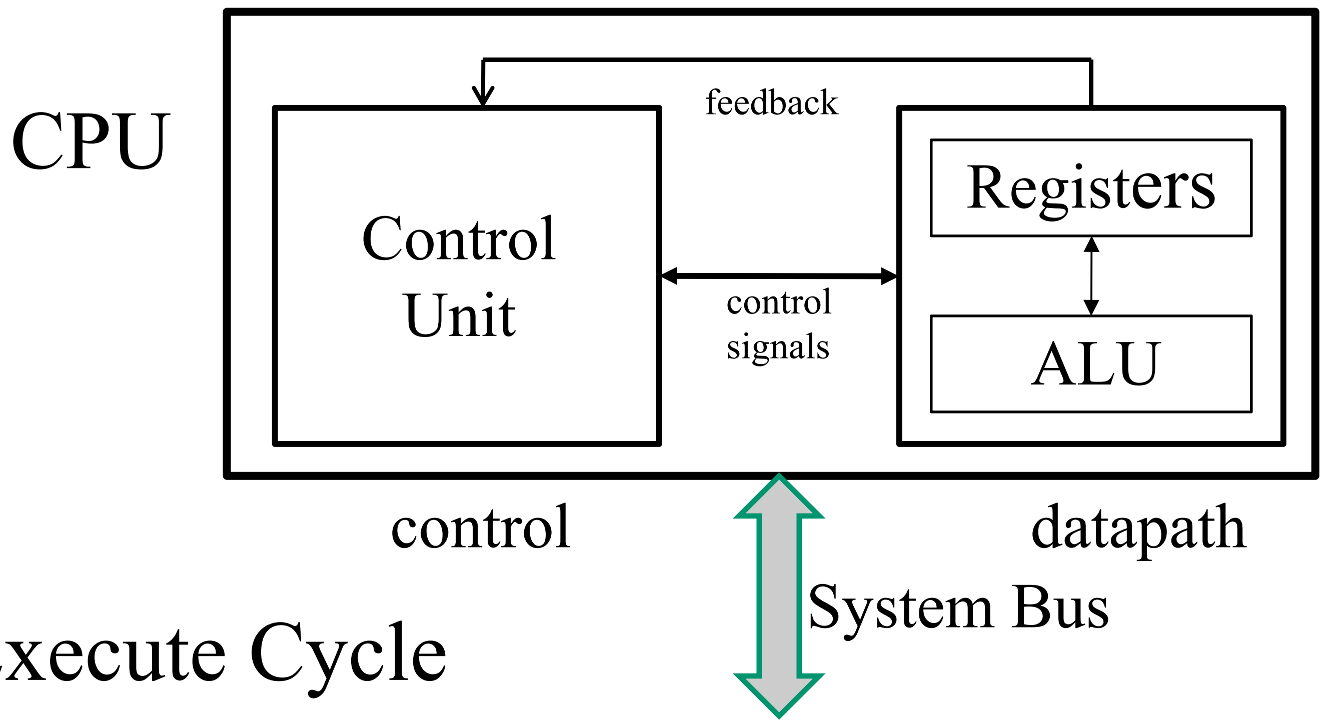
What architecture is the RX600 ?



CPU Components

- The Arithmetic Logic Unit (ALU)
  - Performs the arithmetic and logic operations
  - It receives data from main memory and/or registers
  - It stores the results back into memory and/or registers
- Registers: are used to hold data and address values.
  - The number and type of registers depends on the CPU design
- The Control Unit:
  - Fetches machine language instructions from memory
  - Decodes the instruction
  - Controls the interpretation of each instruction including:

    fetching the required operands (data) from registers or memory
    controlling the required executions steps
    storing the results in registers or memory



Fetch-Execute Cycle

The steps that the control unit carries out in executing a program are:

(1) Fetch the next instruction to be executed from memory.

(2) Decode the opcode.

(3) Read operand(s) from main memory, if any.

(4) Execute the instruction and store results, if any.

(5) Go to step 1.

Register Transfer Operations

Consider the 8086 instruction: MOV

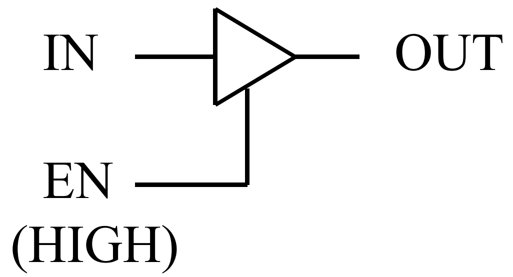
mov Move data
Syntax: mov dest, src
dest: register or memory
src: register, memory, or
immediate Action: dest = src
Flags Affected: None

In the datapath, data is held in registers.

MOV C,B $C \leftarrow B$

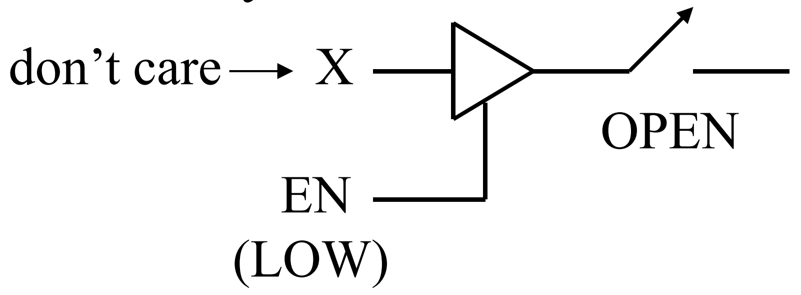
Tristate Buffer - Reminder

A tristate gate has three output states, HIGH, LOW, high-impedance (high-Z).



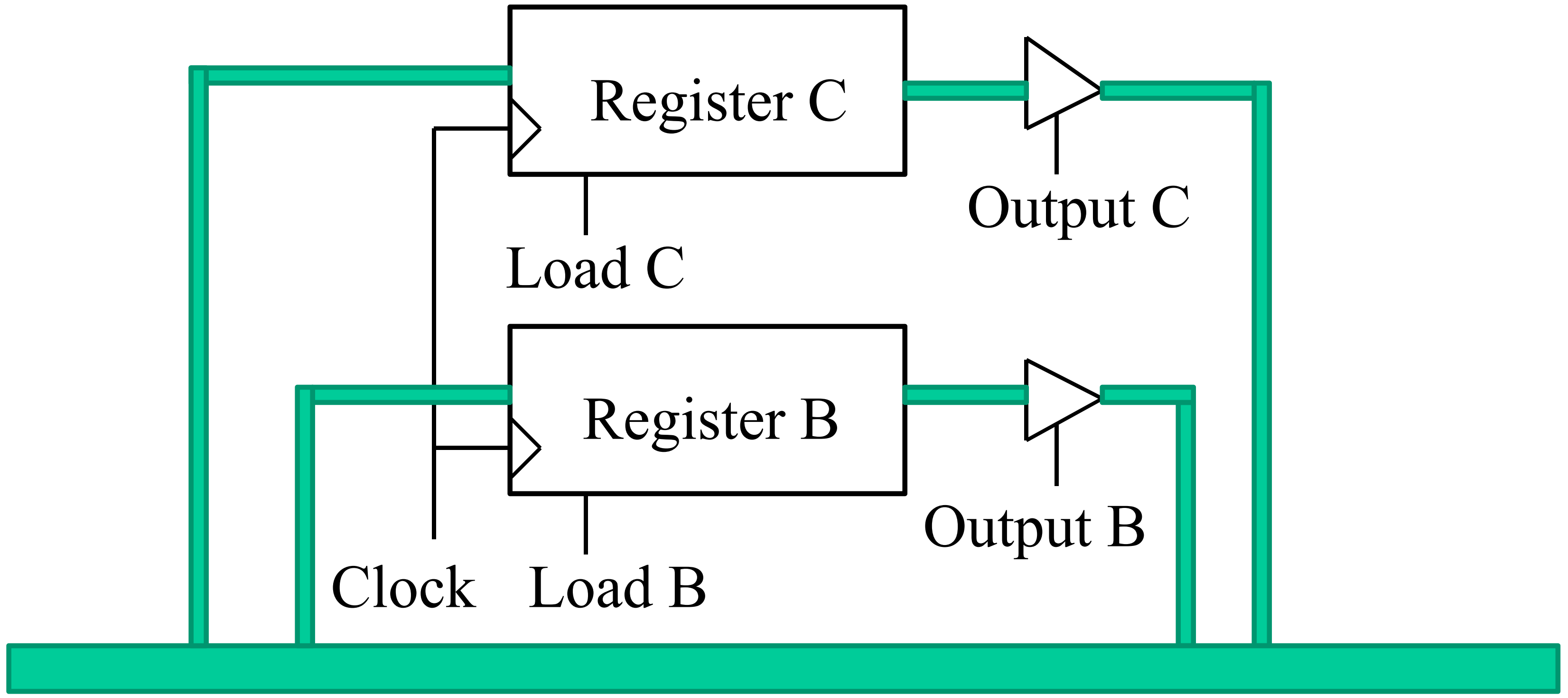
When enable (EN) is HIGH, the gate will function as normal, in this case, passing the input to the output. Setting enable LOW selects high-Z operation. The output is effectively disconnected from the rest of the circuit by internal circuitry.

This allows us to connect outputs together which may otherwise destroy the gates.



Datapath with Registers

D-Type flip-flops, synchronous, controlled by a LOAD signal.



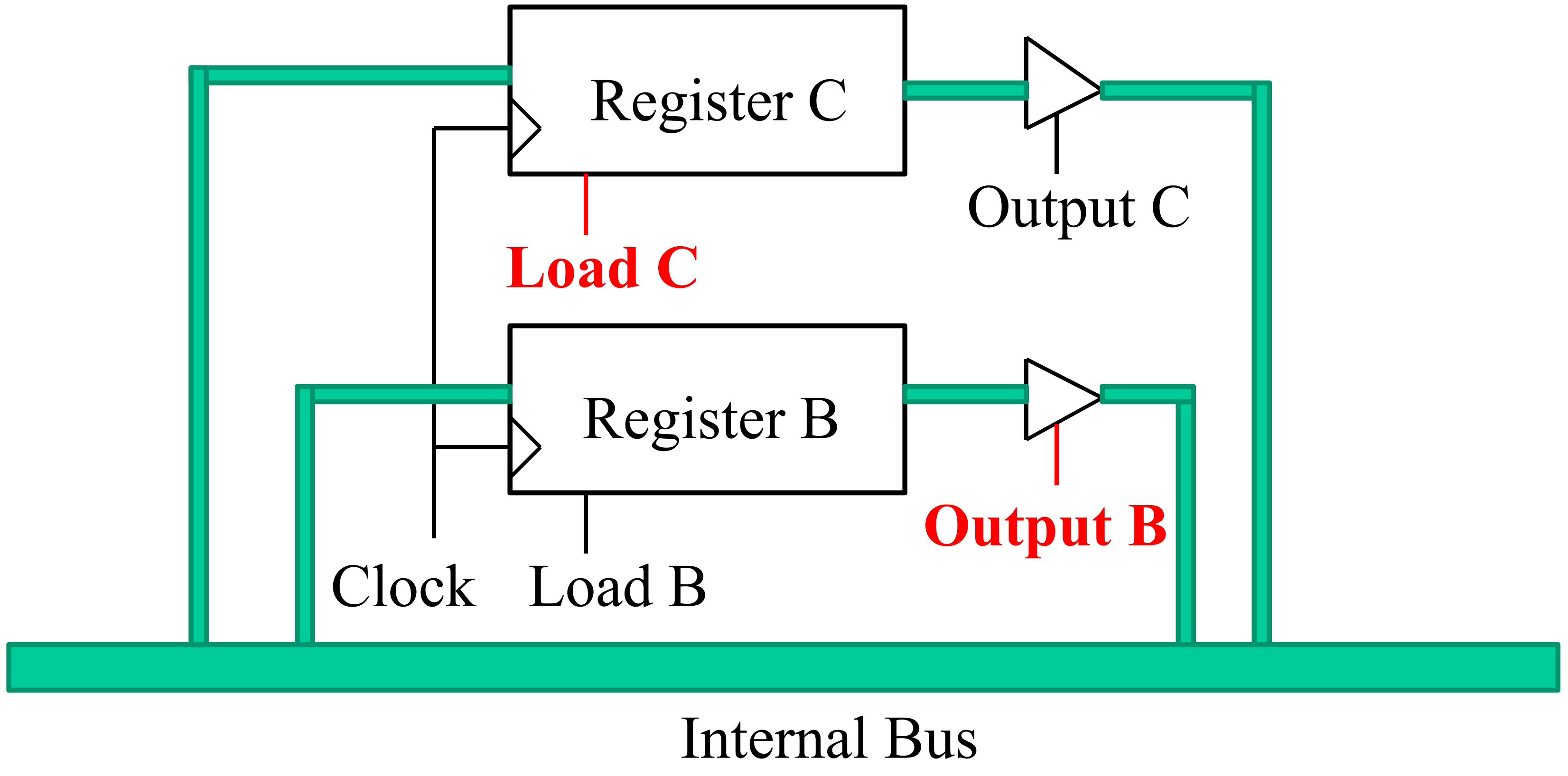
Internal Bus

MOV C, B

1. Assert ‘Output B’ and ‘Load C’
2. Clock into destination register.

Datapath with Registers

D-Type flip-flops, synchronous, controlled by a LOAD signal.

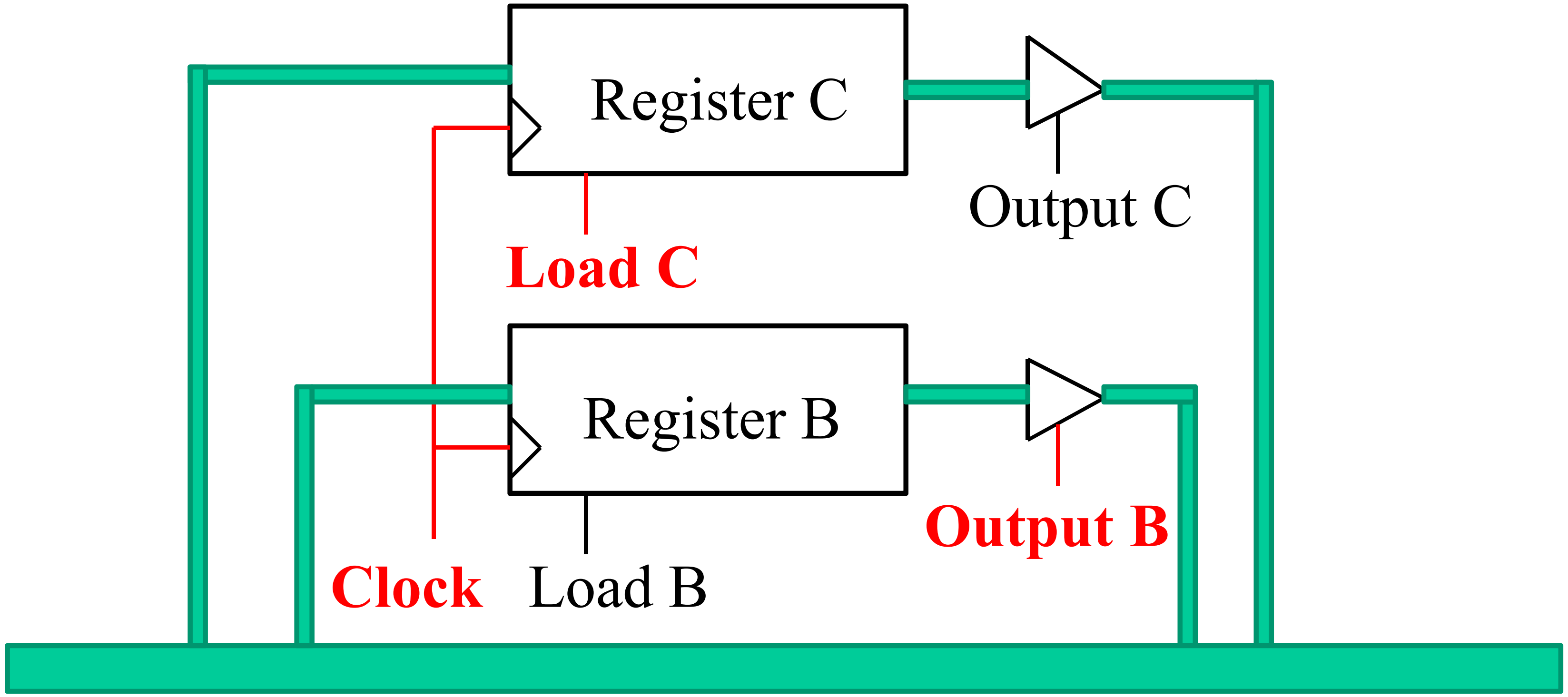


MOV C, B

1. **Assert ‘Output B’ and ‘Load C’**
2. Clock into destination register.

Datapath with Registers

D-Type flip-flops, synchronous, controlled by a LOAD signal.



Internal Bus

MOV C, B

1. Assert ‘Output B’ and ‘Load C’
2. **Clock into destination register.**

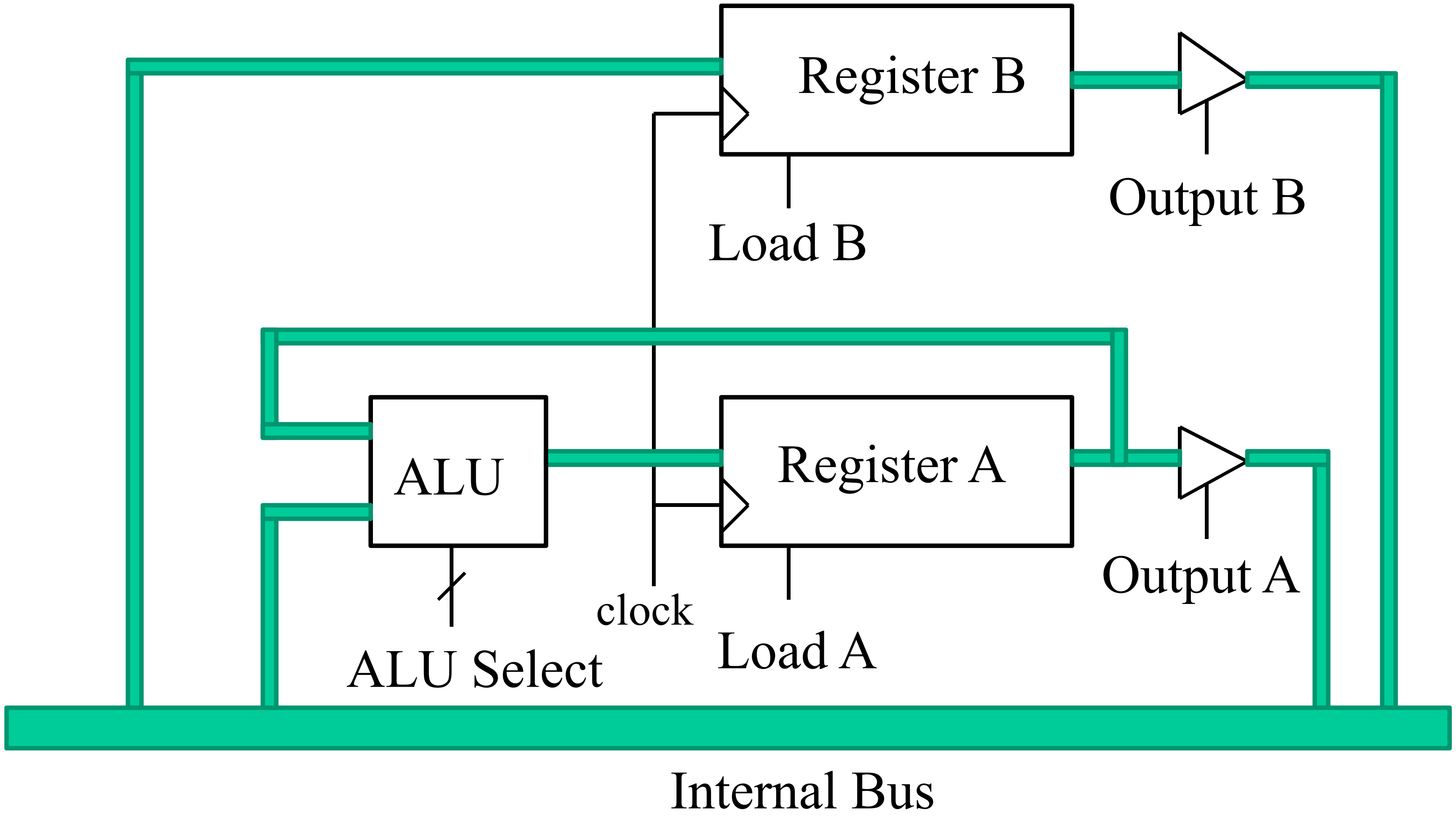
Addition

Consider the 8086 instruction: ADD

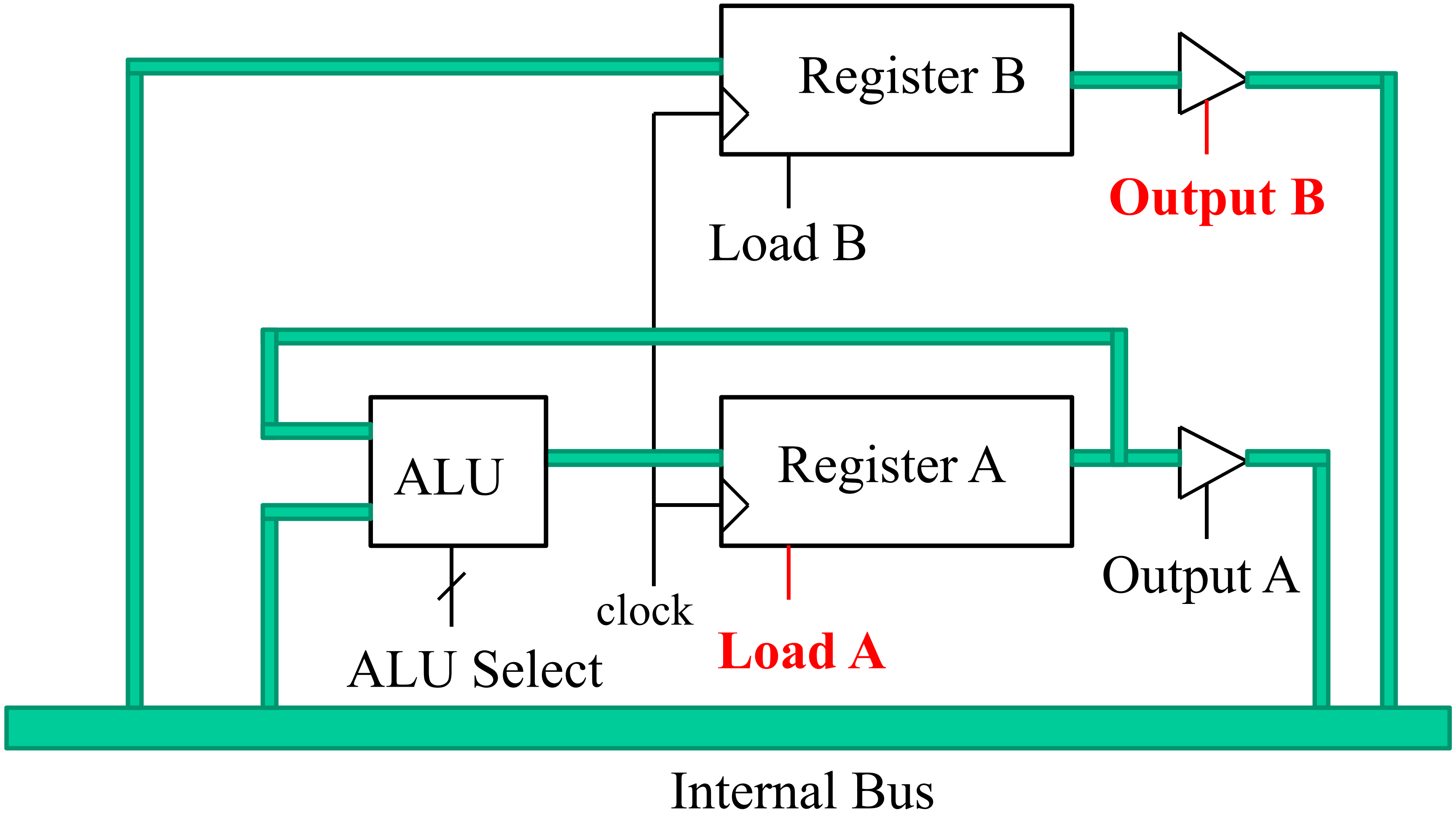
Add   Add two numbers Syntax: add dest, src
dest: register or memory
src: register, memory, or immediate Action: dest =
dest + src
Flags Affected: OF, SF, ZF, AF, PF, CF
Notes: Works for both signed and unsigned numbers.

ADD A,B $A \leftarrow A + B$

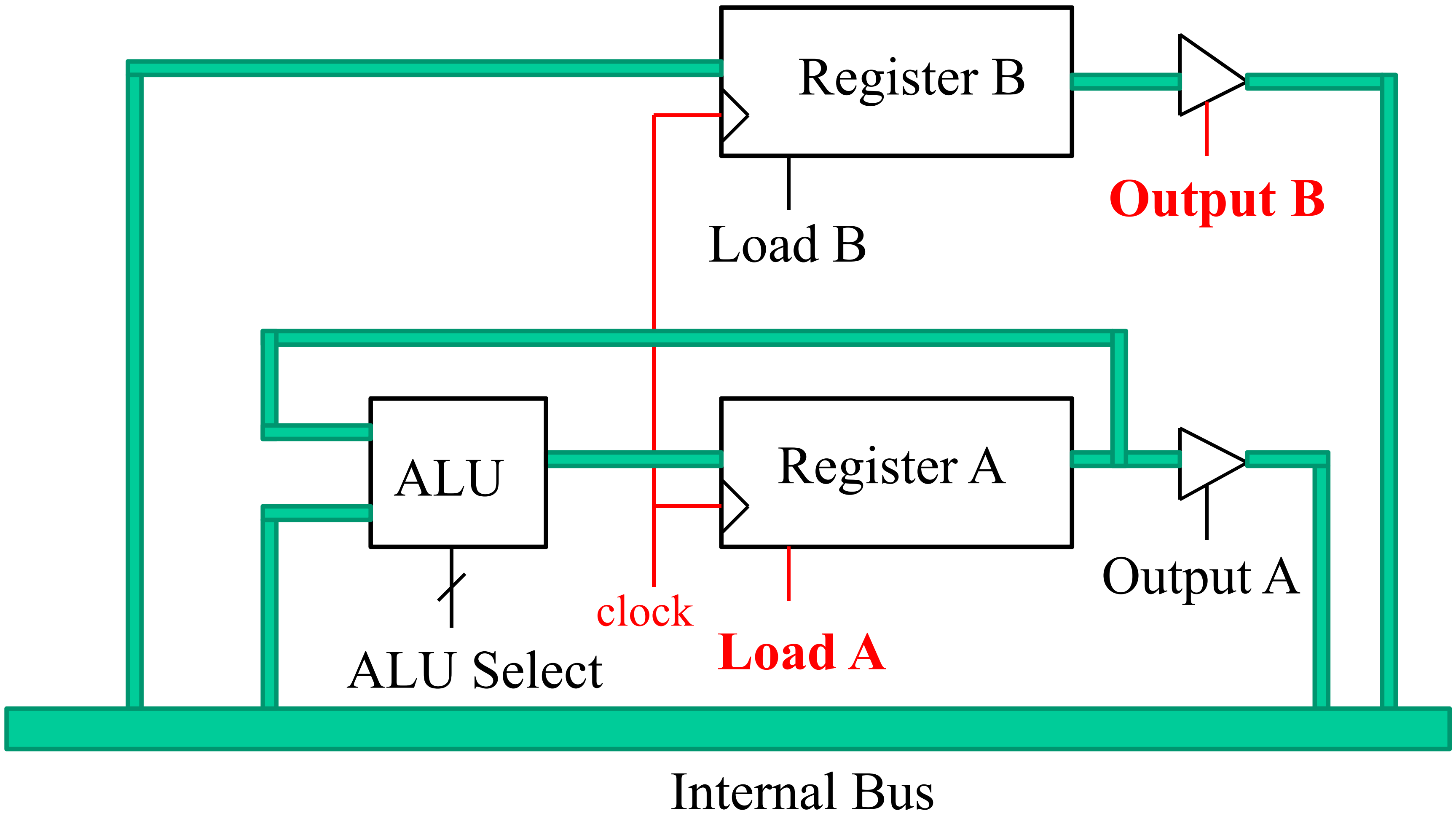
Datapath with Registers and ALU



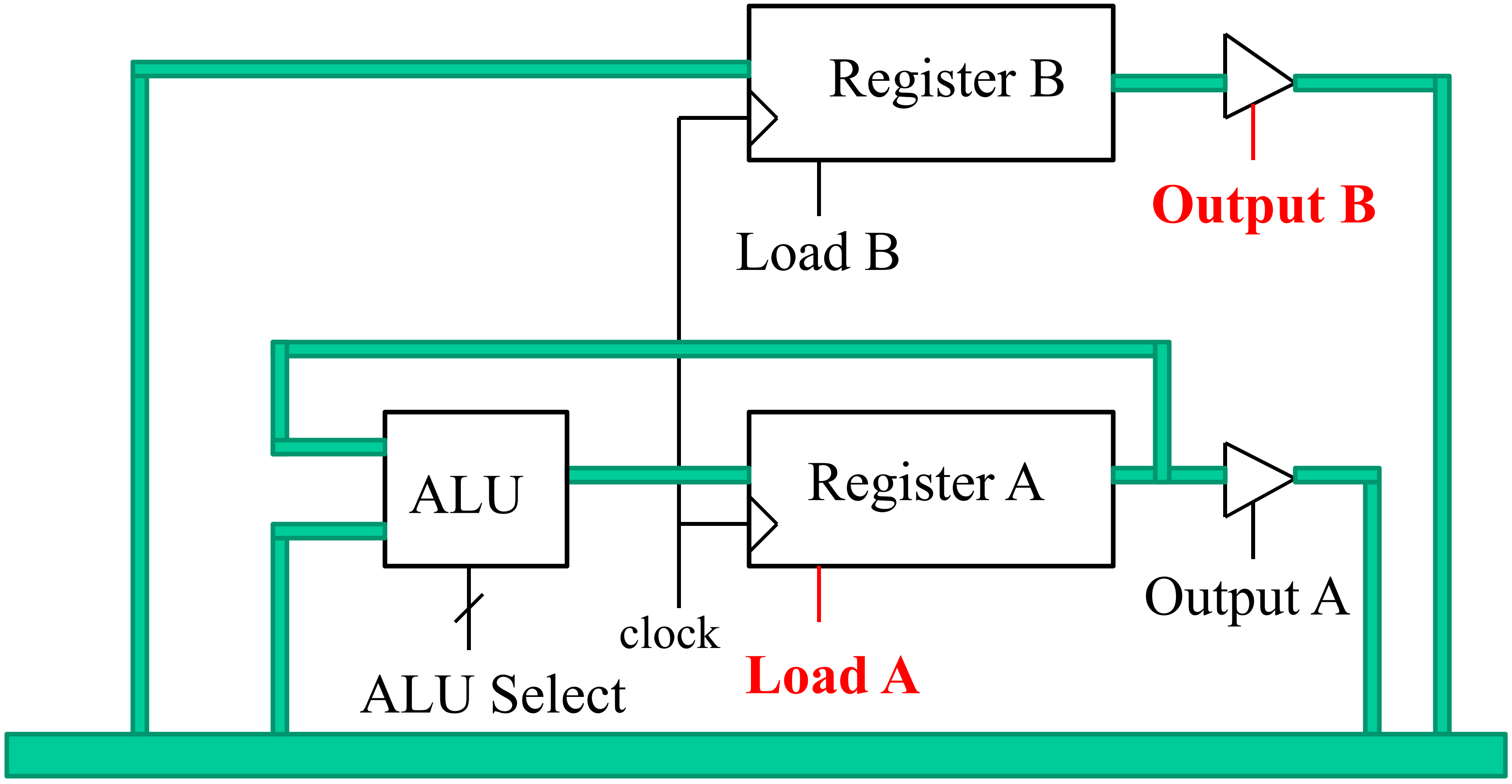
Datapath with Registers and ALU



Datapath with Registers and ALU

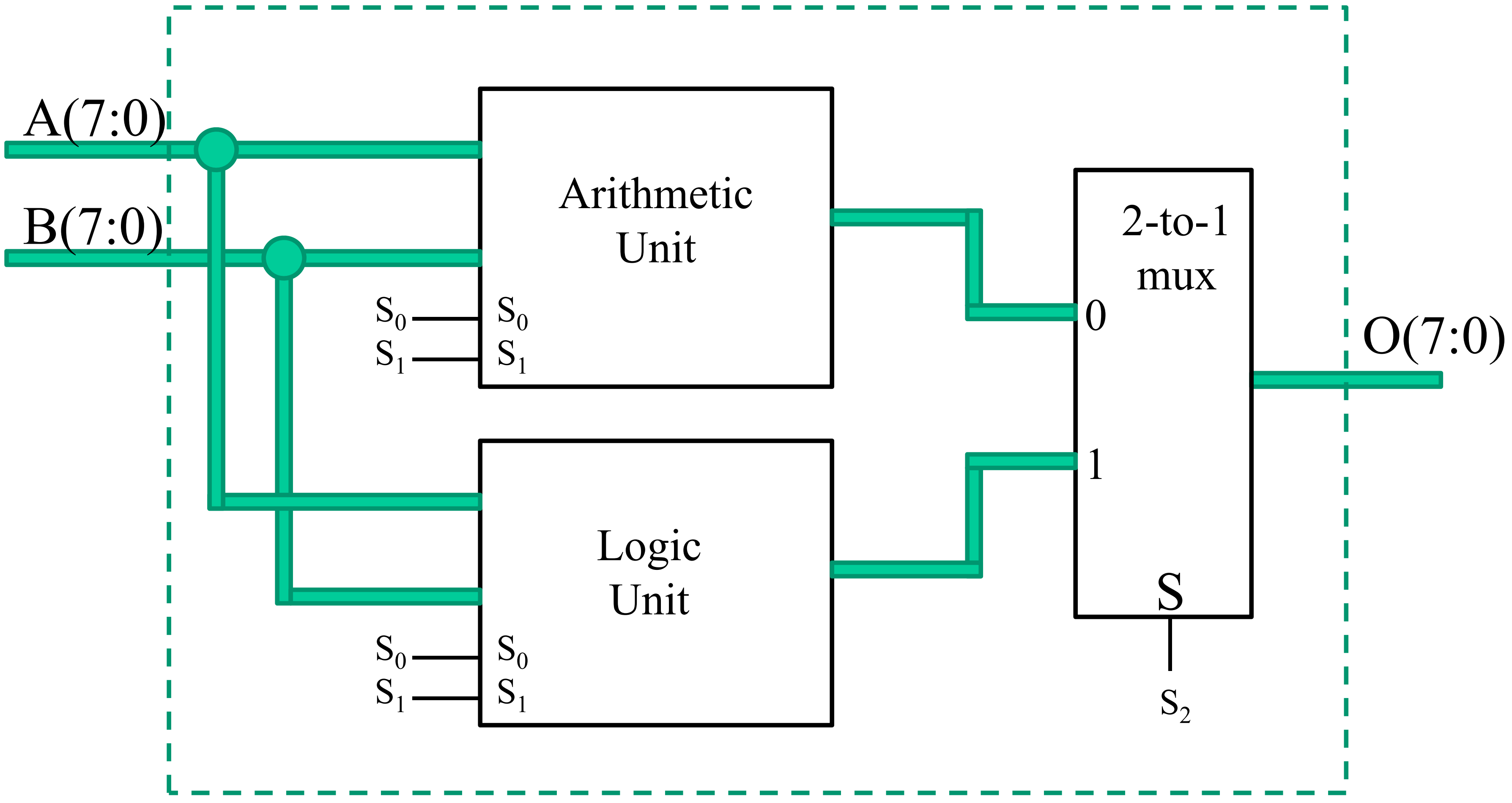


Datapath with Registers and ALU



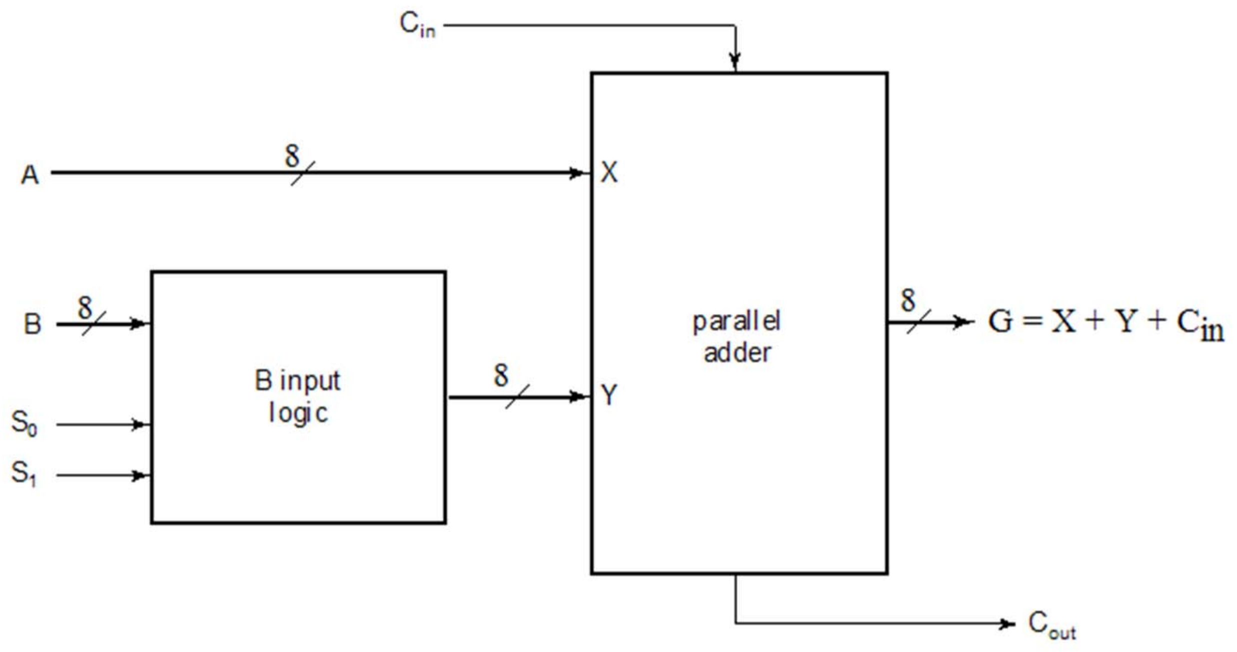
Internal Bus

The ALU



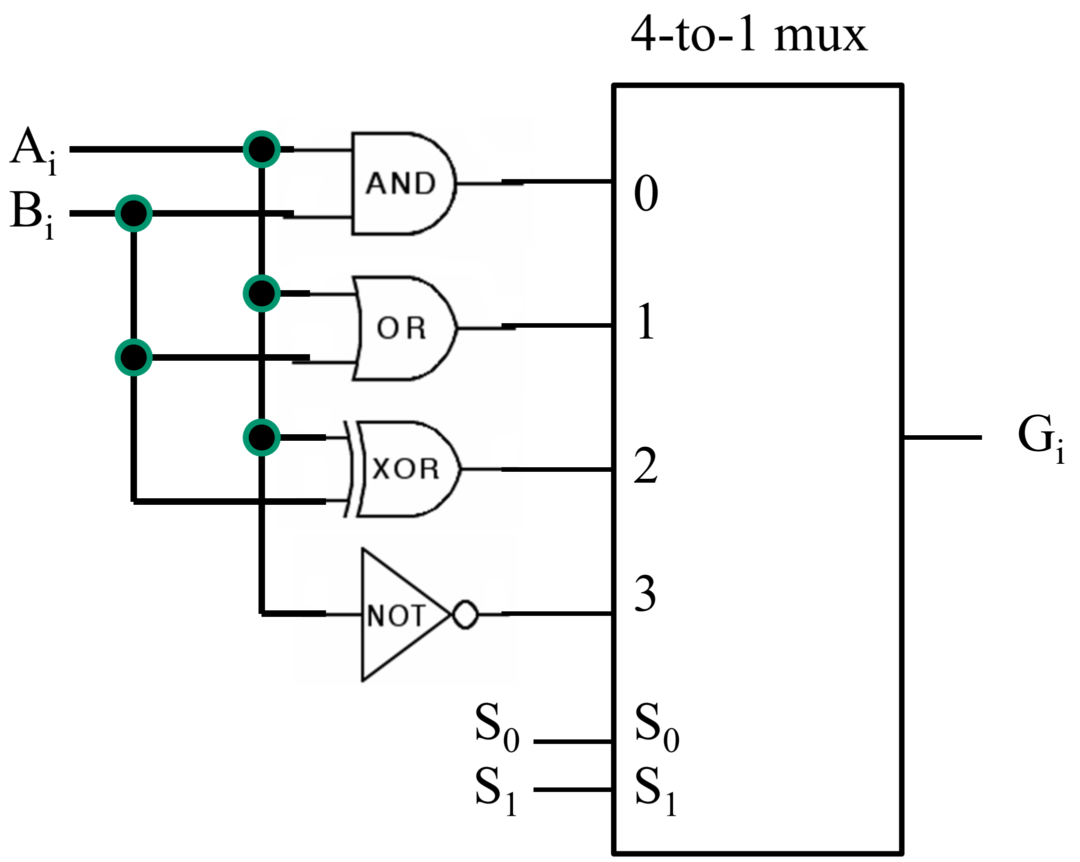
A(7:0)
B(7:0)
Arithmetic Unit
S_0
S_1
S_0
S_1
Logic Unit
S_0
S_1
S_0
S_1
2-to-1 mux
0
1
S
S_2
O(7:0)

Arithmetic Unit



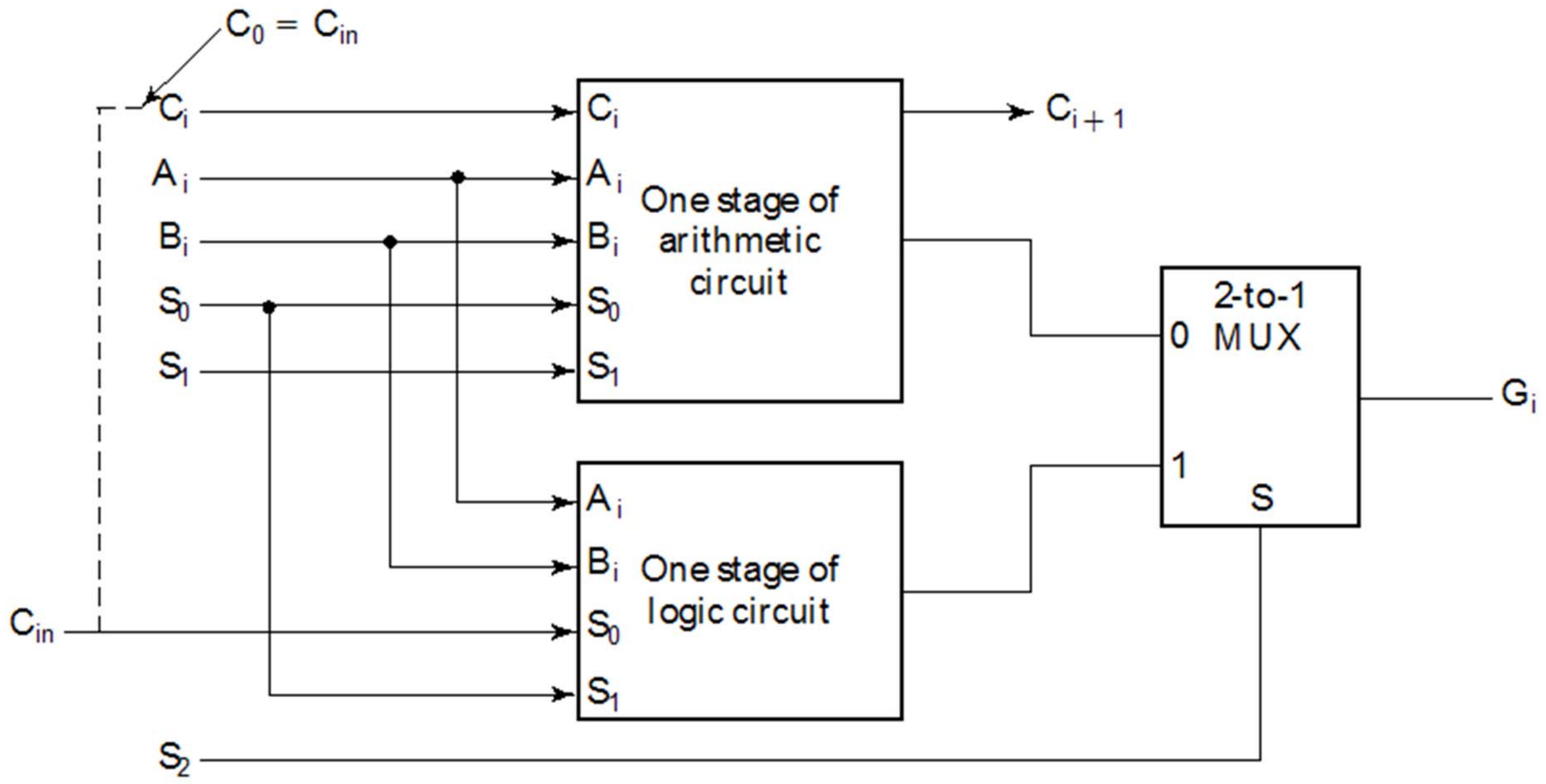
Select		Input	G = X + Y + Cin	
S1	S0	Y	Cin=0	Cin = 1
0	0	all 0's	G = A (transfer)	G = A + 1 (increment)
0	1	B	G = A + B (add)	G = A + B + 1
1	0	$\overline{B}$	G = A + $\overline{B}$	G = A + $\overline{B}$ + 1 (subtract)
1	1	all 1's	G = A – 1 (decrement)	G = A (transfer)

One Stage Logic Circuit (bitslice)



S1	S0	Operation
0	0	AND
0	1	OR
1	0	XOR
1	1	NOT

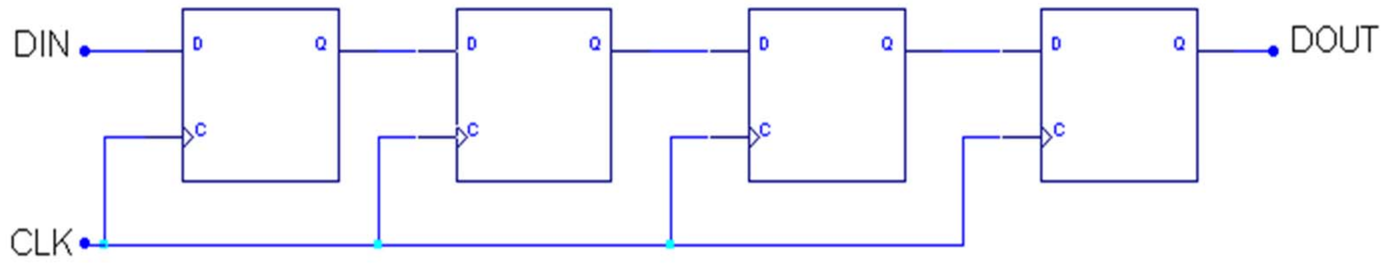
One Stage of ALU



The method used to drive the select lines in this example was chosen to help with instruction decoding. It is equally valid to have three direct inputs for the three select lines.

The Shifter

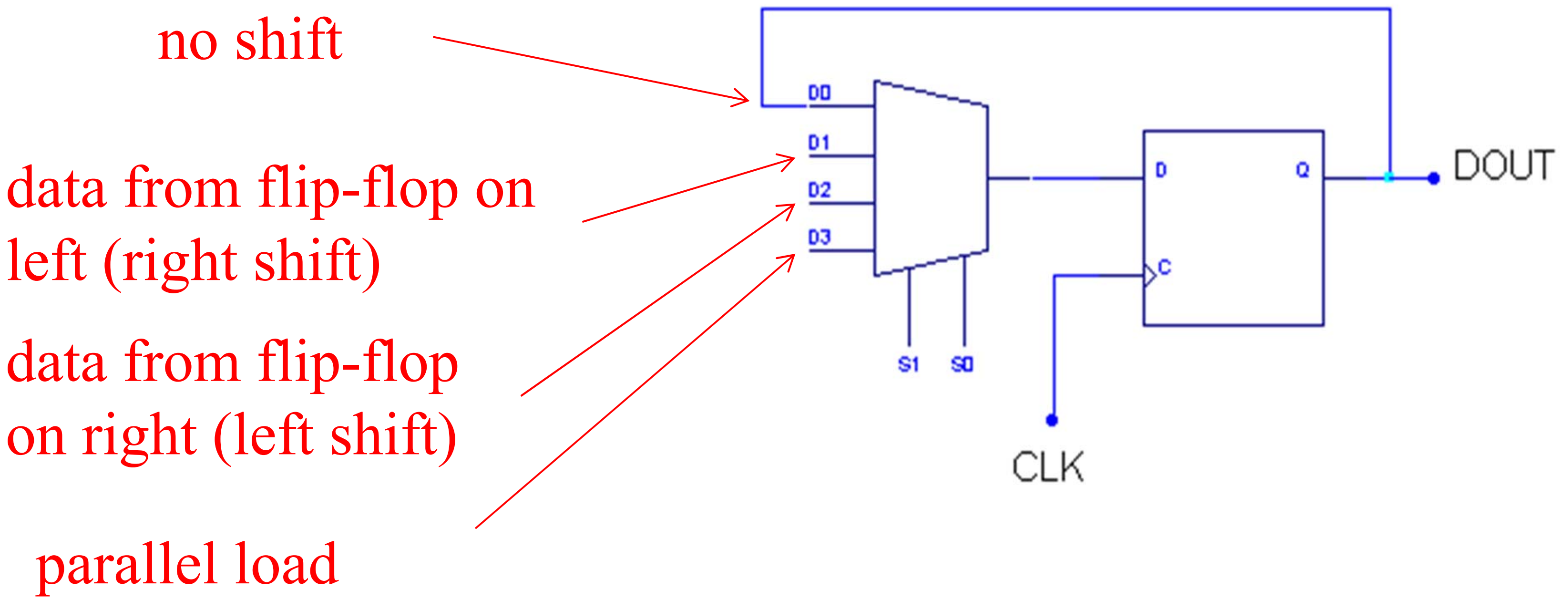
The obvious choice for the datapath shifter is to use a standard bidirectional shift register based on D-Type flip-flops. These are connected so that the output of one flip-flop feeds the input to the next.



A multiplexer on each input gives a choice of left shift, right shift, parallel load and no-shift, by selecting the source of the D input appropriately.

Shifter

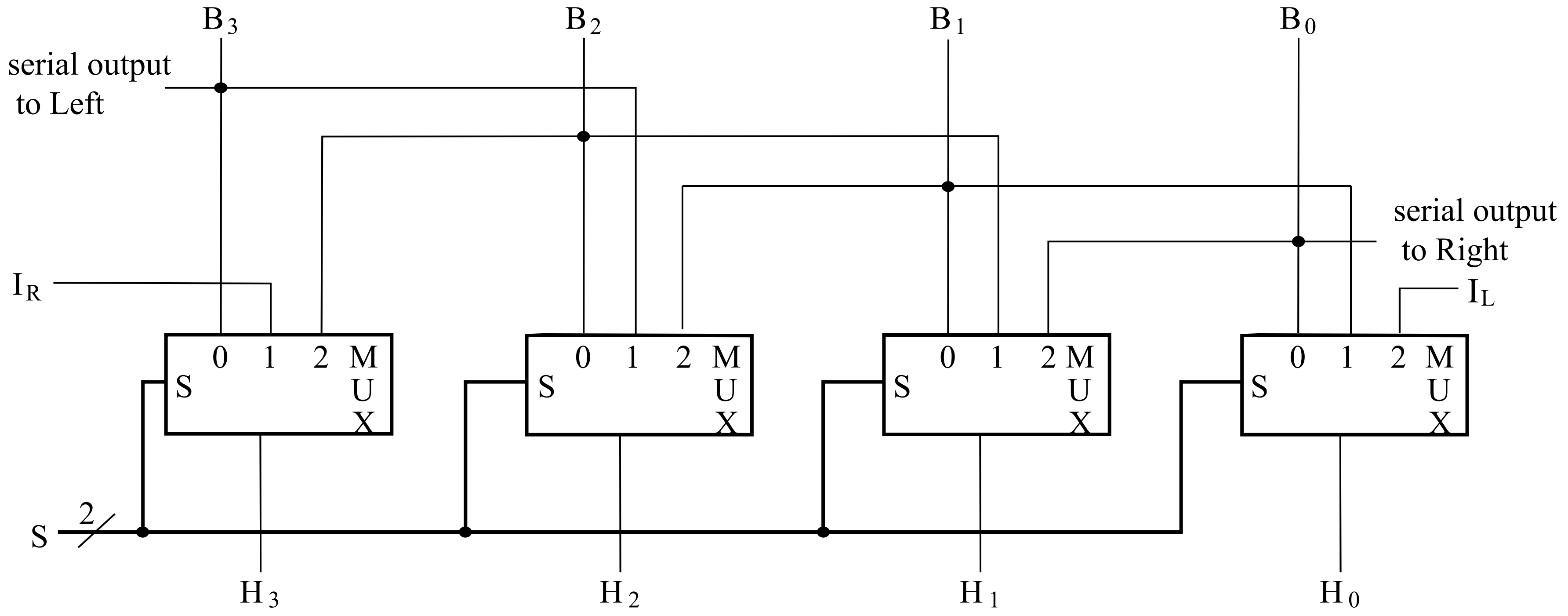
A multiplexer on each input gives left shift, right shift, parallel load and unchanged. One possible configuration is indicated in red.


no shift
data from flip-flop on left (right shift)
data from flip-flop on right (left shift)
parallel load
D0
D1
D2
D3
S1
S0
D
Q
C
DOUT
CLK

However, this uses three clock cycles. One to load the shifter from the source register, one to shift, one to load the destination register.

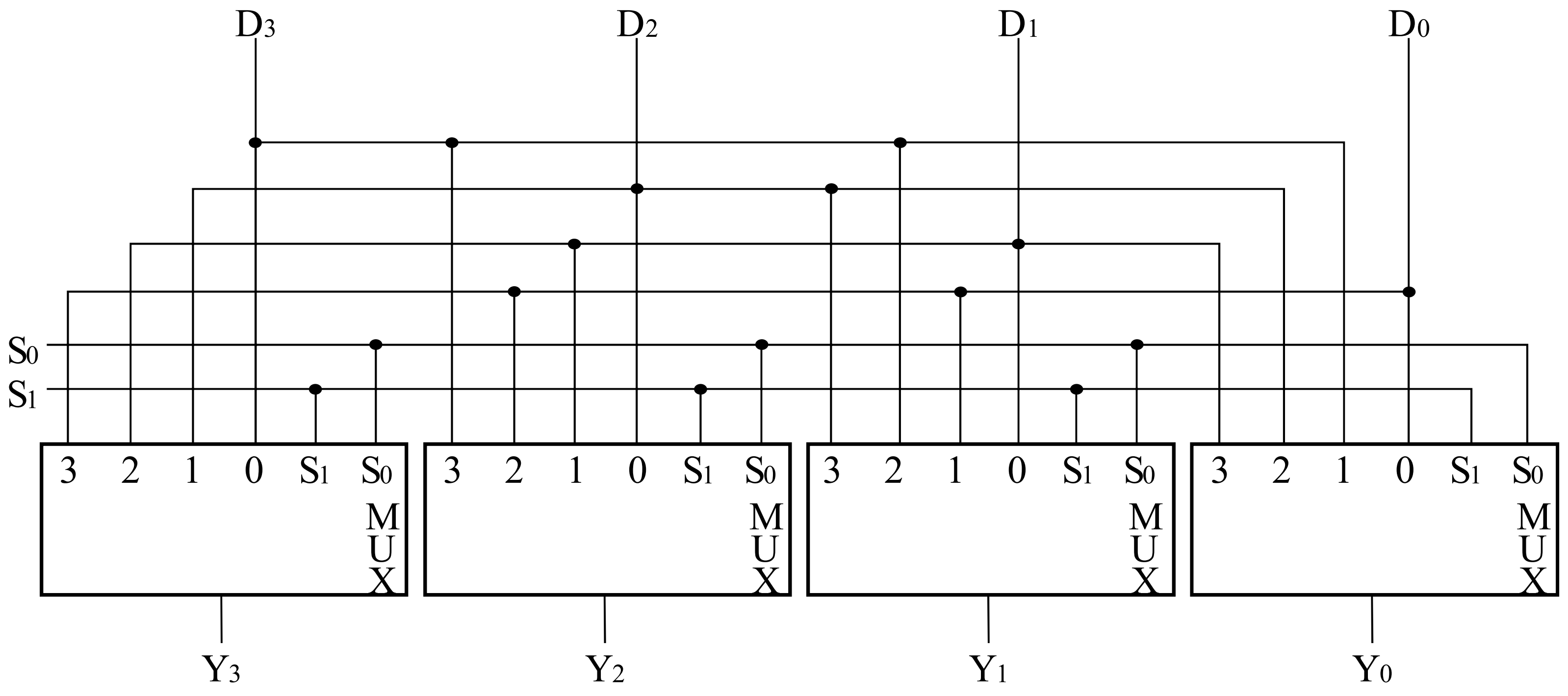
Combinational Shifter

A faster solution implements the shifter as a combinational circuit using multiplexers.



B is from the source register output and H is the destination register input. I is serial data from the right or left when increasing bit length.

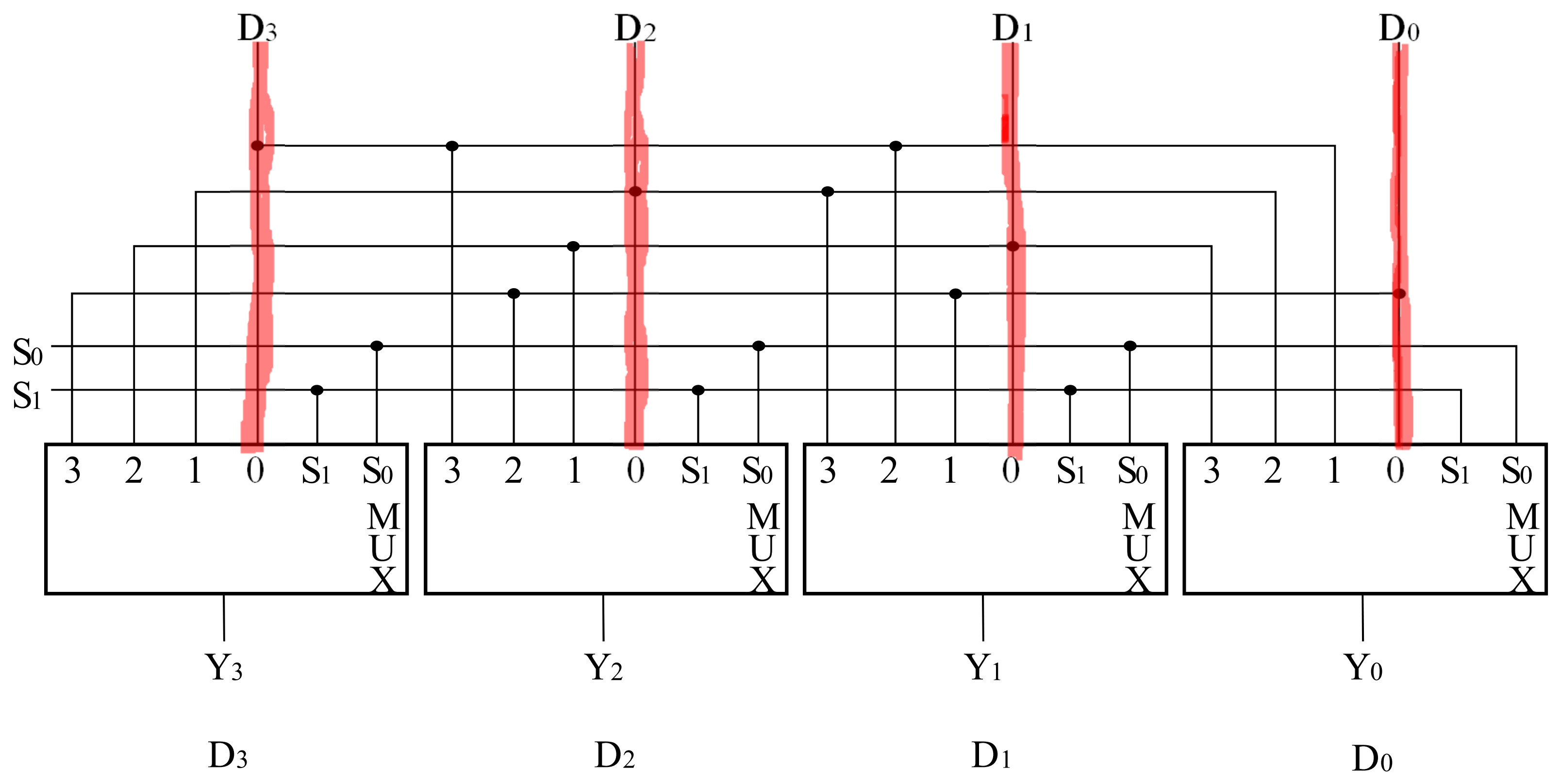
Barrel Shifter



A rotate is a shift in which the bits shifted out are inserted into the positions vacated. The circuit rotates its contents left from 0 to 3 positions depending on S:

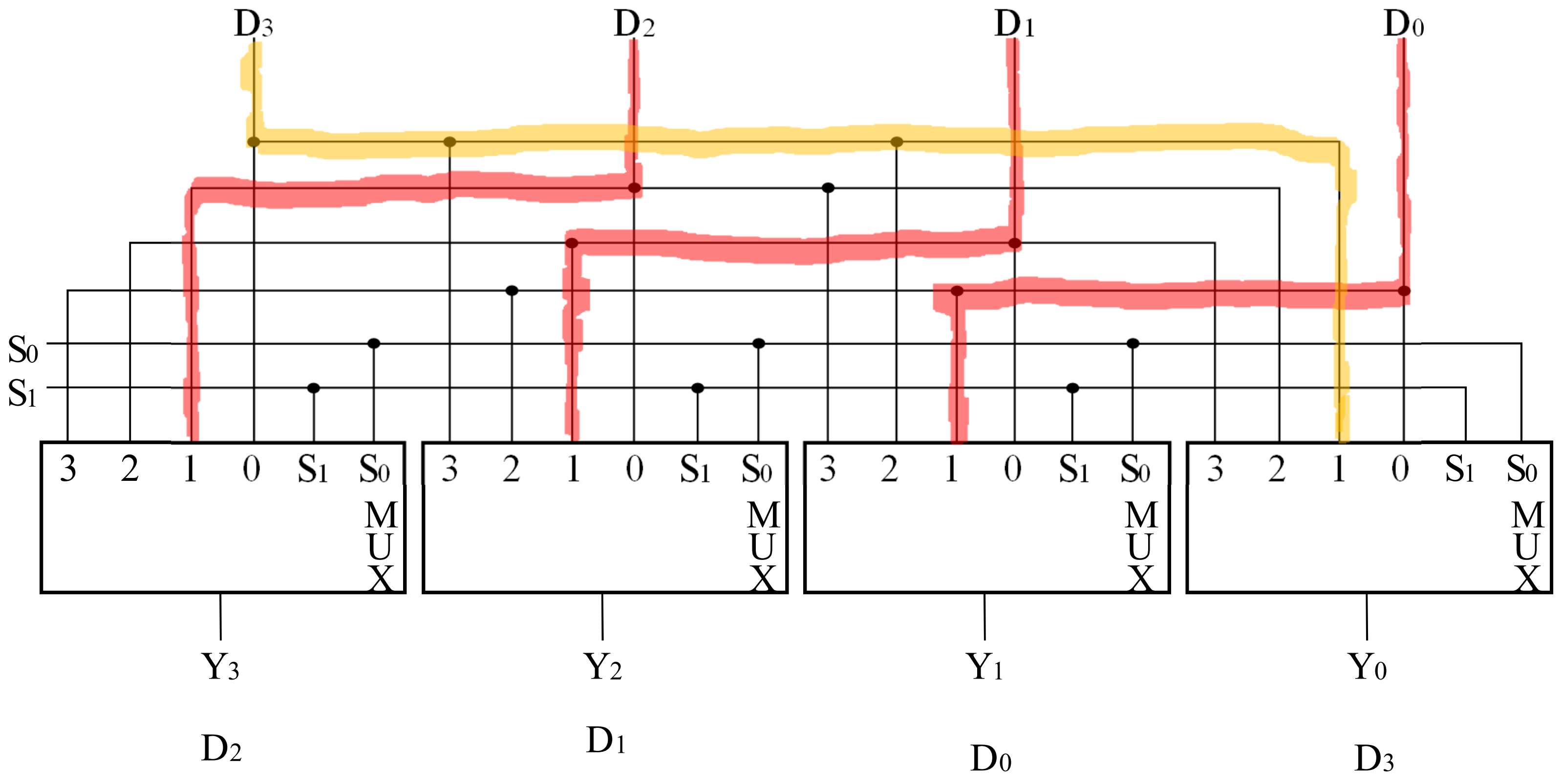
S = 00 position unchanged
S = 01 rotate left by 1 positions
S = 10 rotate left by 2 positions
S = 11 rotate left by 3 positions

Barrel Shifter



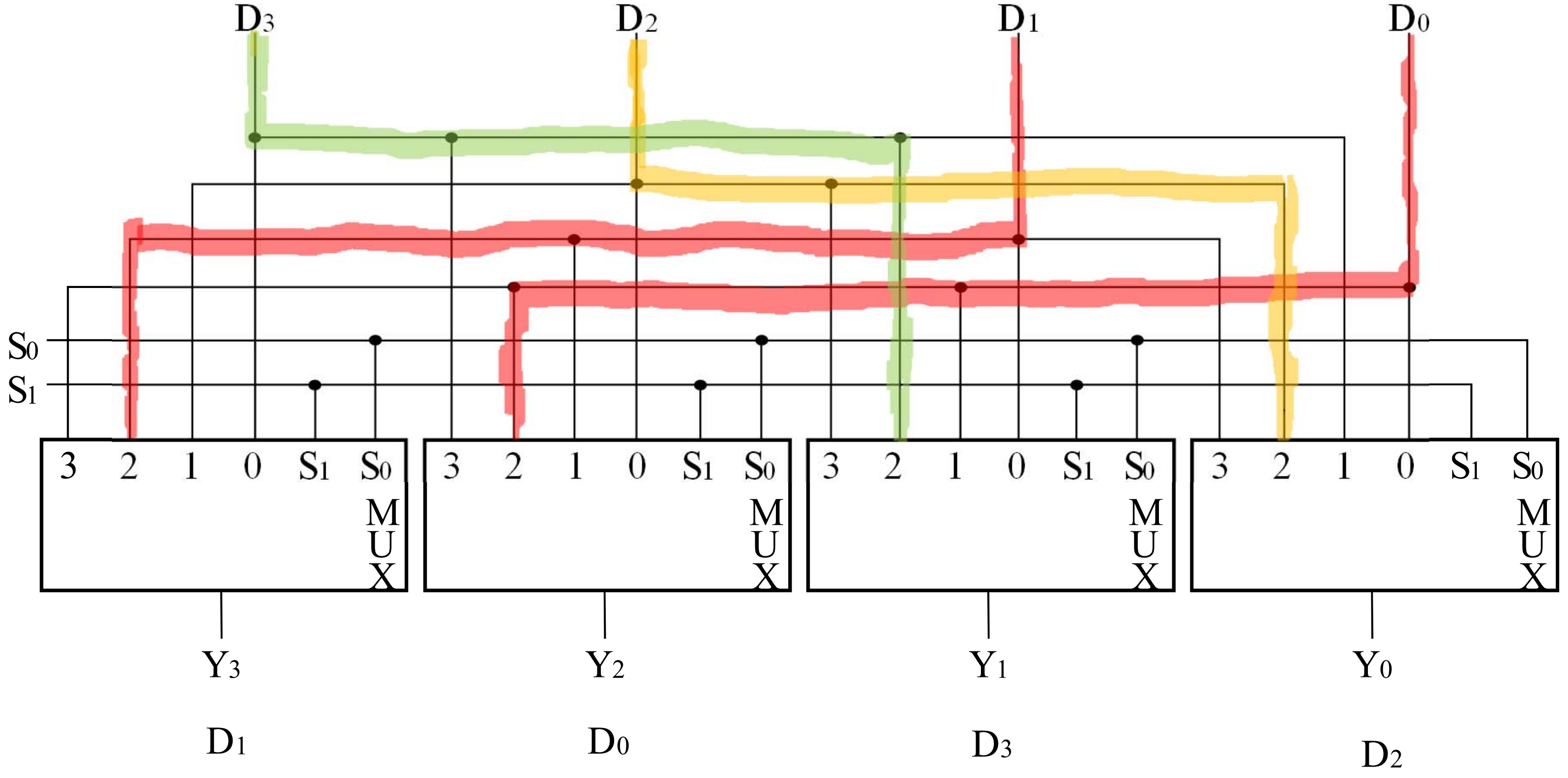
S = 00 position unchanged

Barrel Shifter



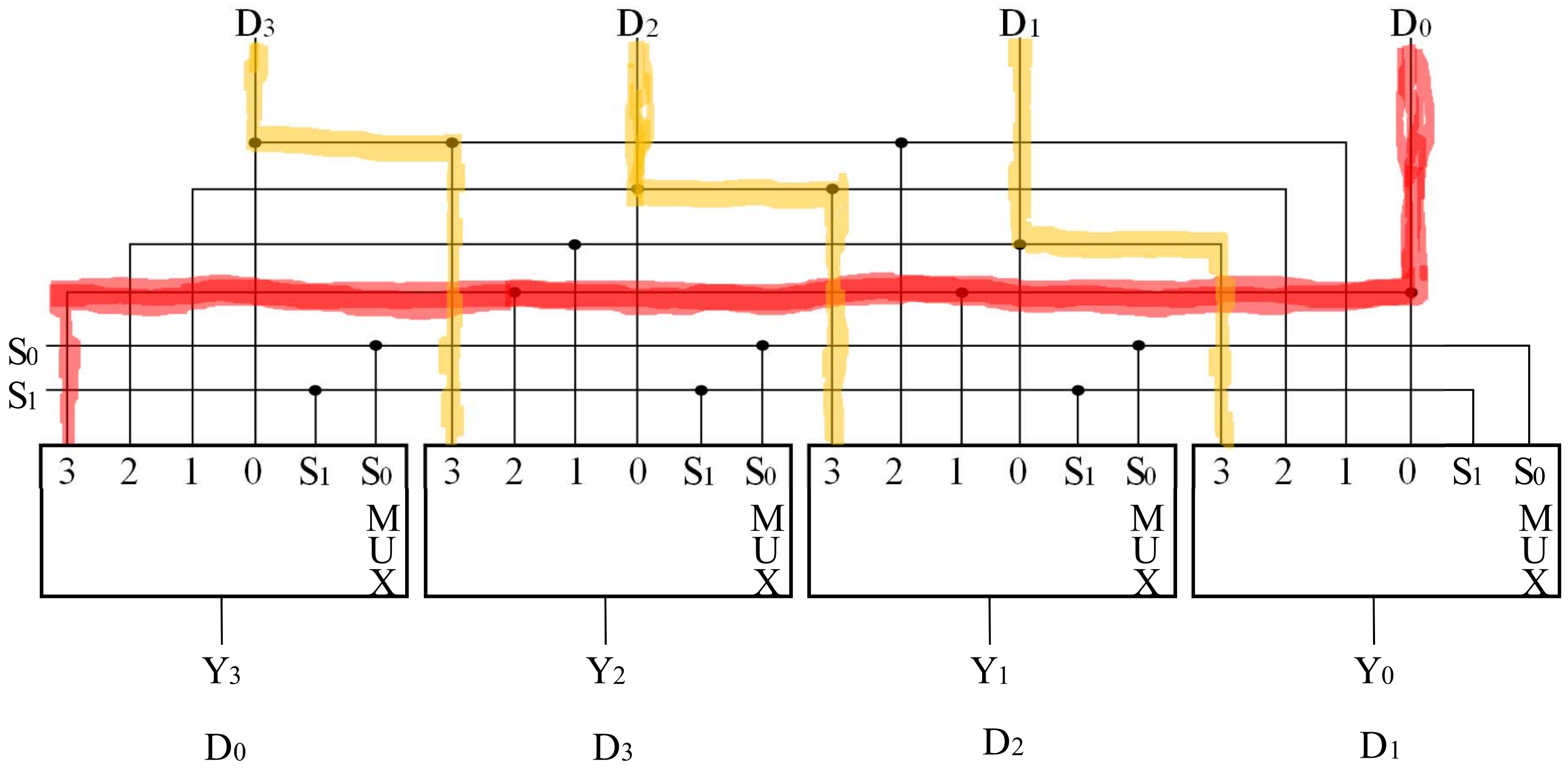
S = 01 rotate left by 1 positions

Barrel Shifter



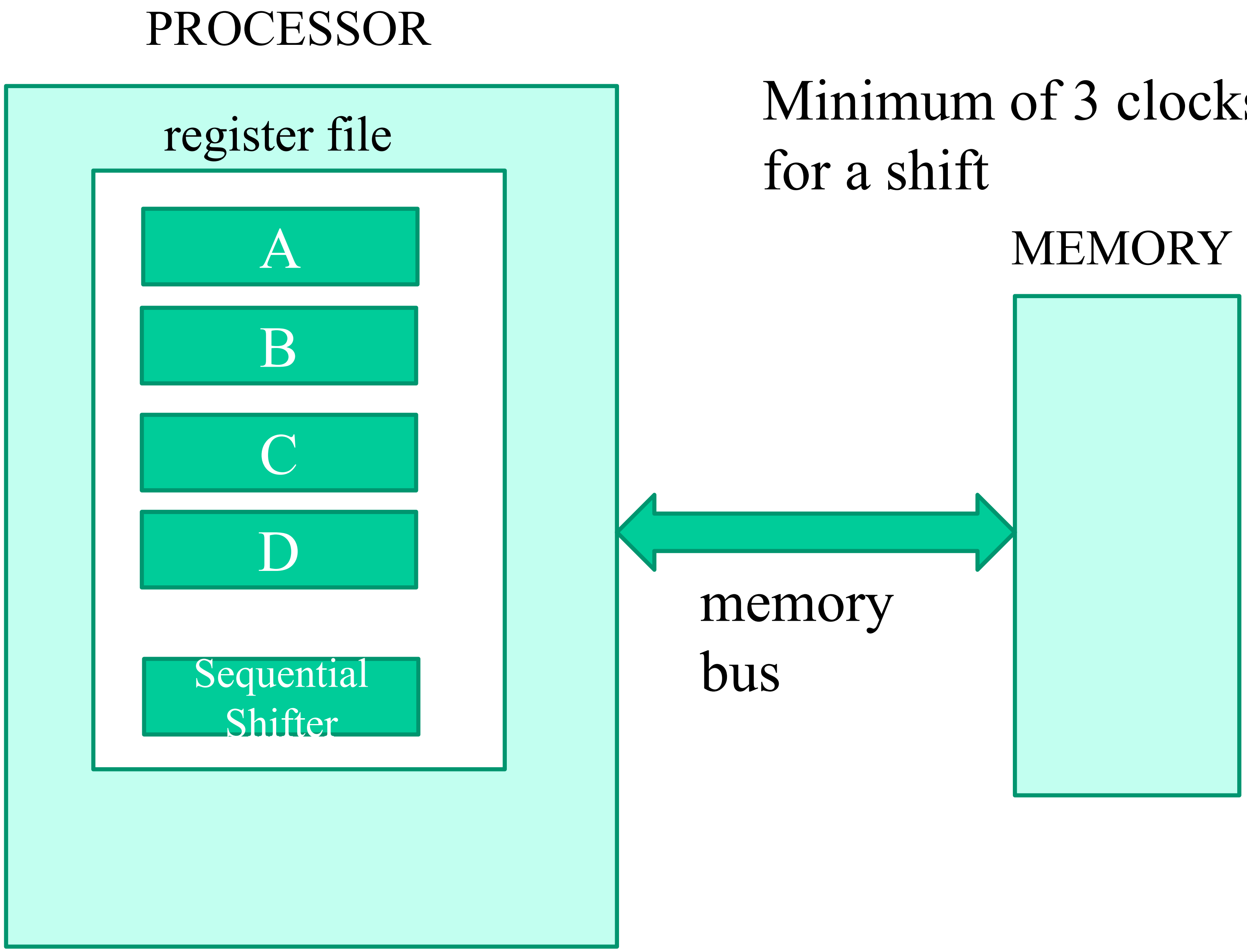
S = 10 rotate left by 2 positions

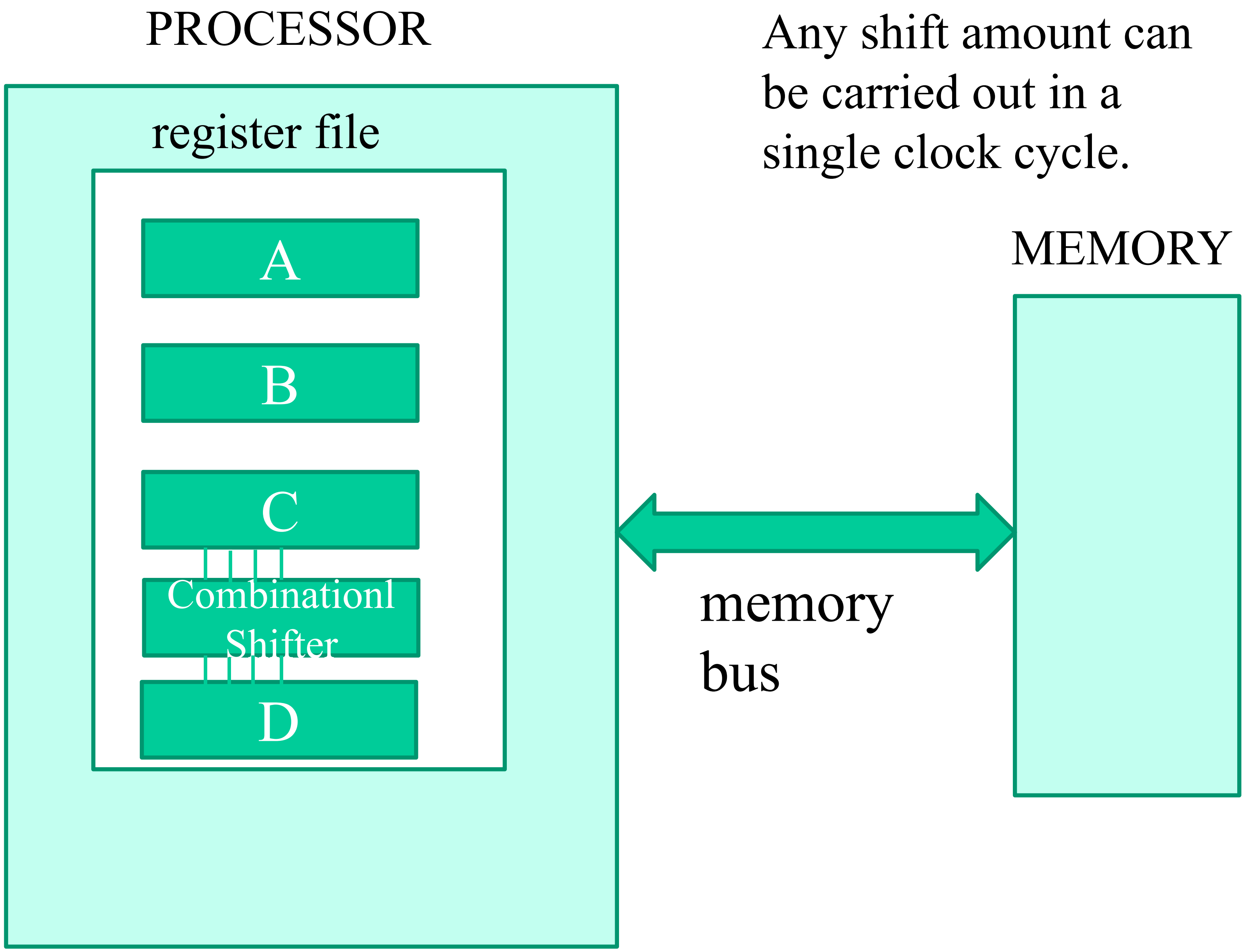
Barrel Shifter



S = 11 rotate left by 3 positions

Minimum of 3 clocks for a shift





Any shift amount can be carried out in a single clock cycle.