A First Look at Microprocessors

using the

The General Prototype Computer (GPC) model

Part 1

The plan:

Present a hypothetical “Generic Prototype Computer” (GPC) model, with features and functions that are found in many/most processors. The GPC exists only as an educational concept, but it could be built and it would work.

After the main concepts have been developed on this simplified model, examine the ARM processor and show how a particular architecture chose to implement the general features and functions.

Along the way, contrast and compare how other architectures have implement the same general features and functions.

A Microprocessor Block Diagram

ALU: See EE214 notes

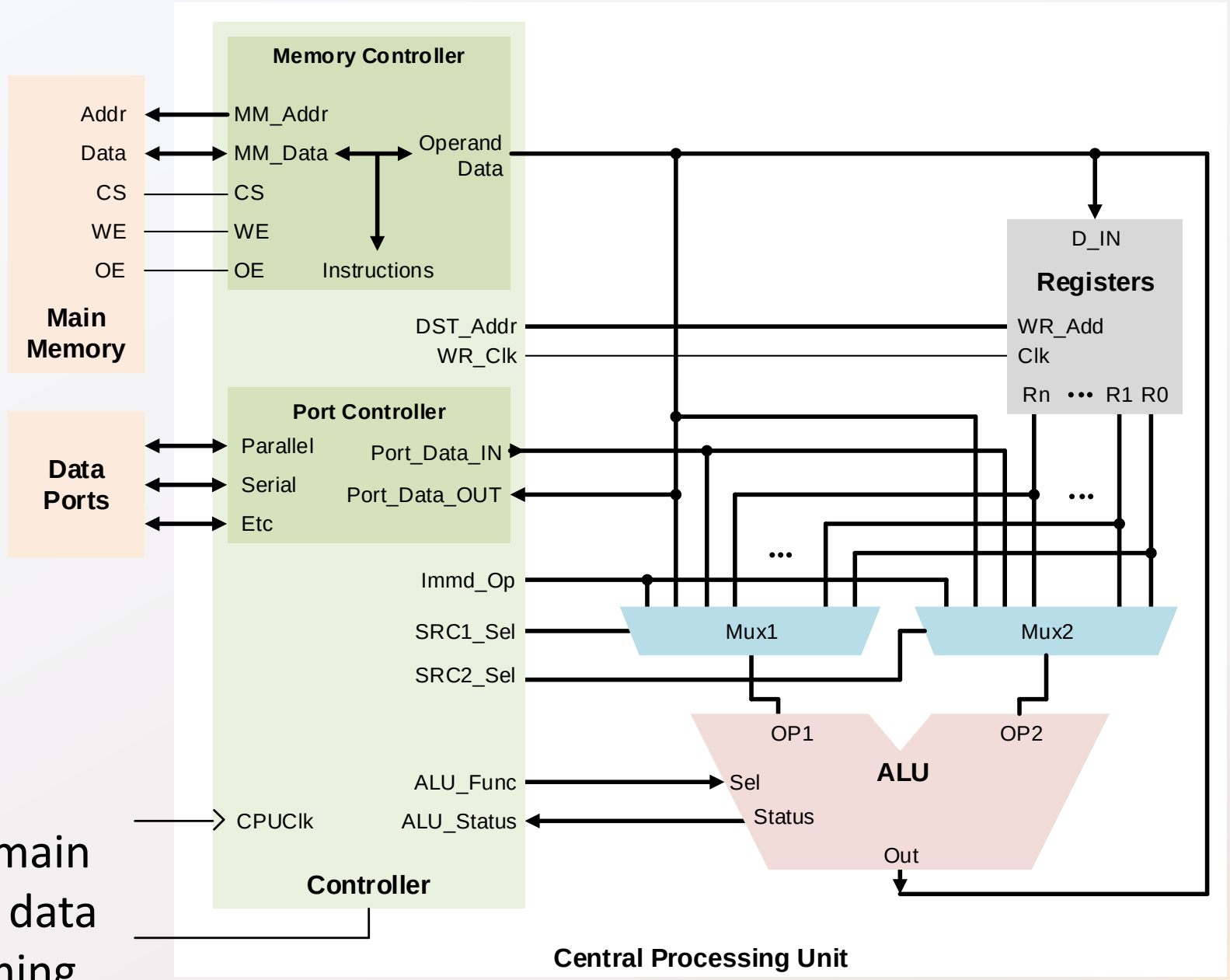
Muxes: Select operands

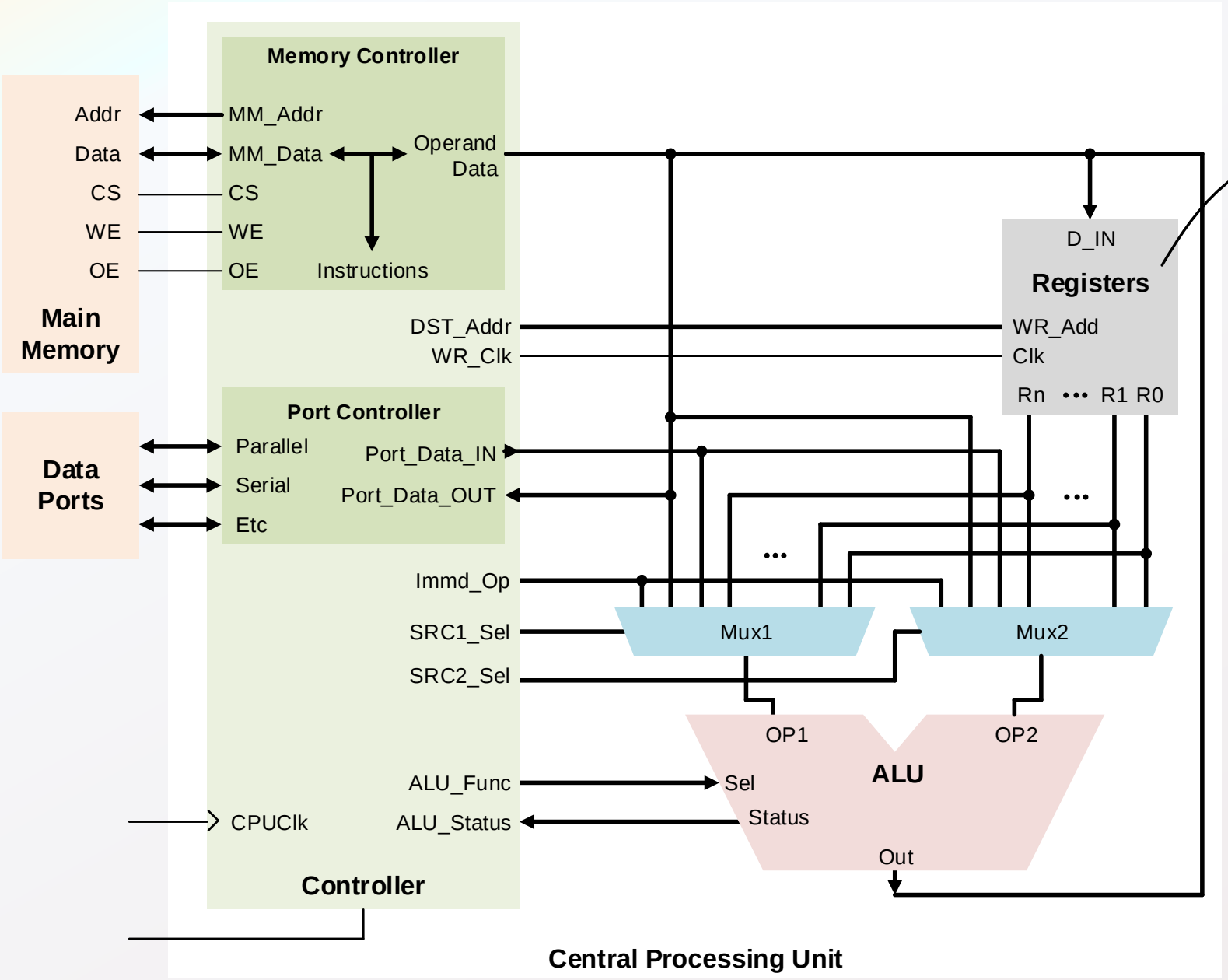
Registers: Temp data storage

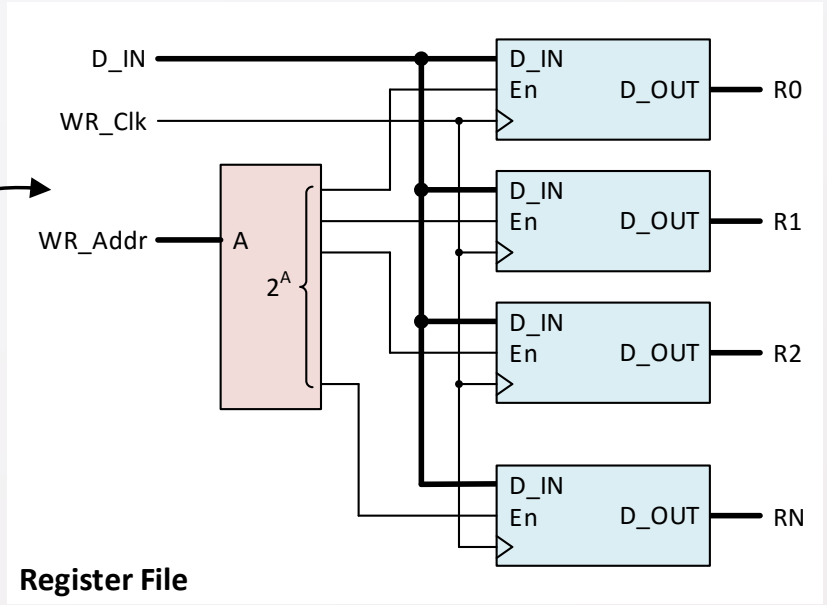
Main Memory: Program and data storage

Ports: Runtime access to the outside world

Controller: “Fetch” opcode from main memory; decode operand; select data sources and destination; drive timing

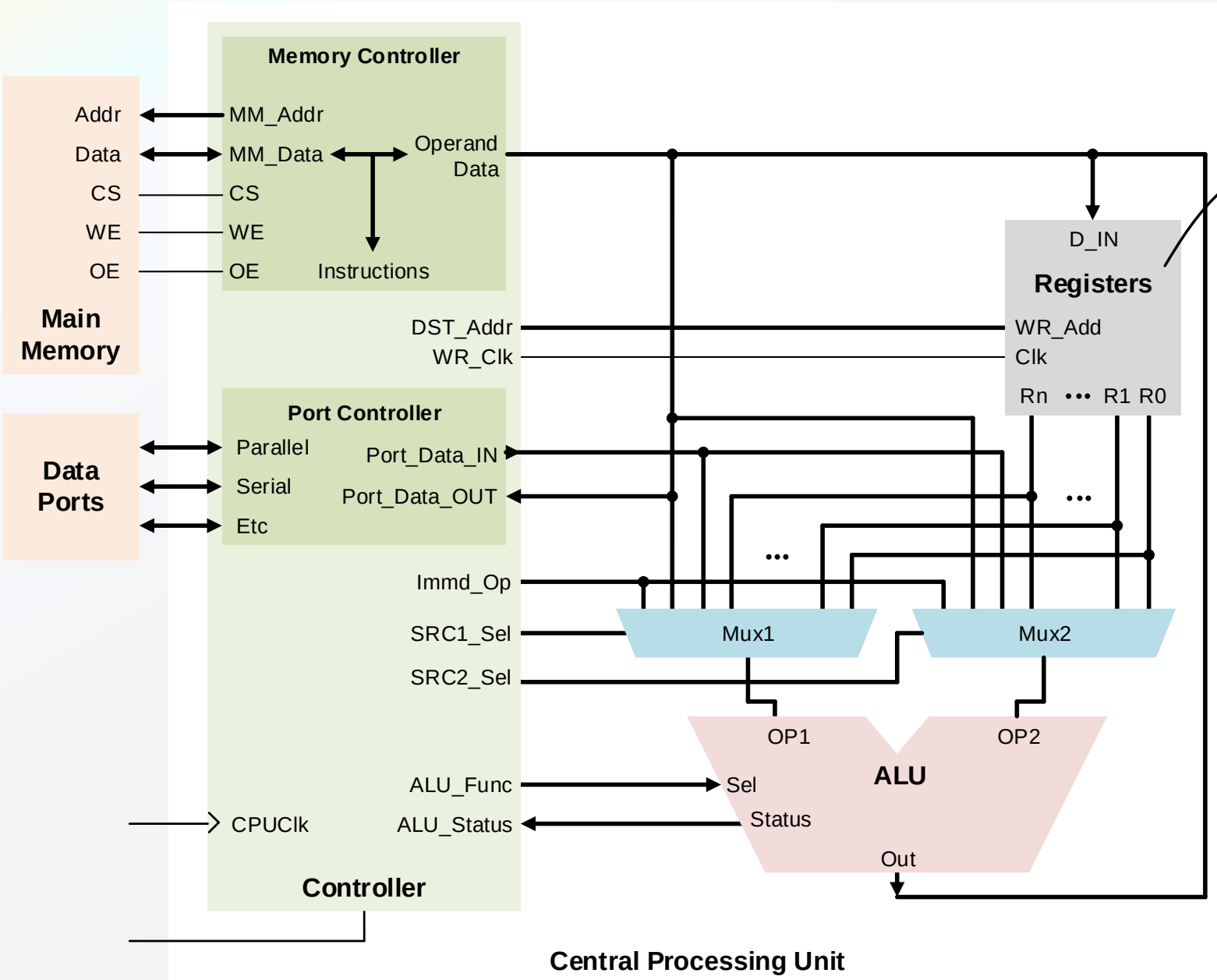


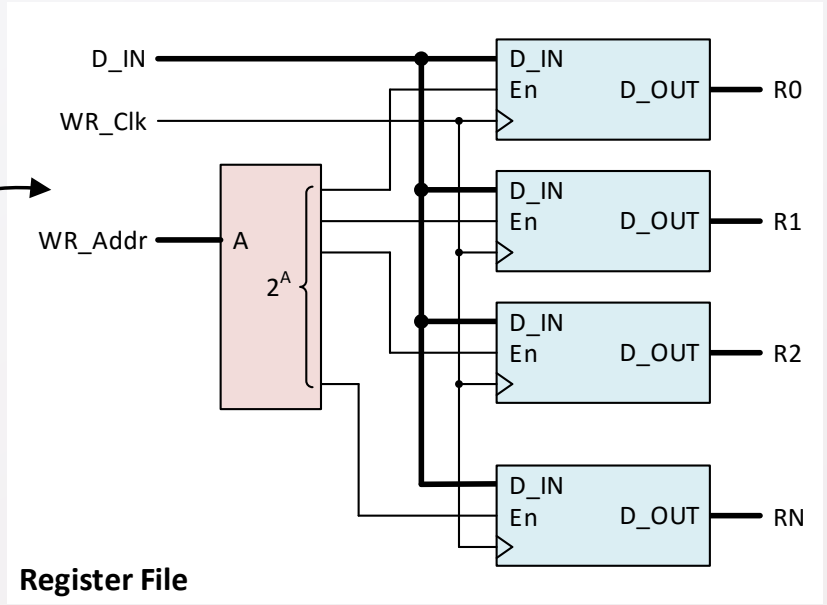




Registers

The Registers are constructed of D flip-flops. They offer high-speed local storage for a small number of data elements. Most CPUs have between 1 and 32 registers. Registers offer the most direct, highest speed access to operands.





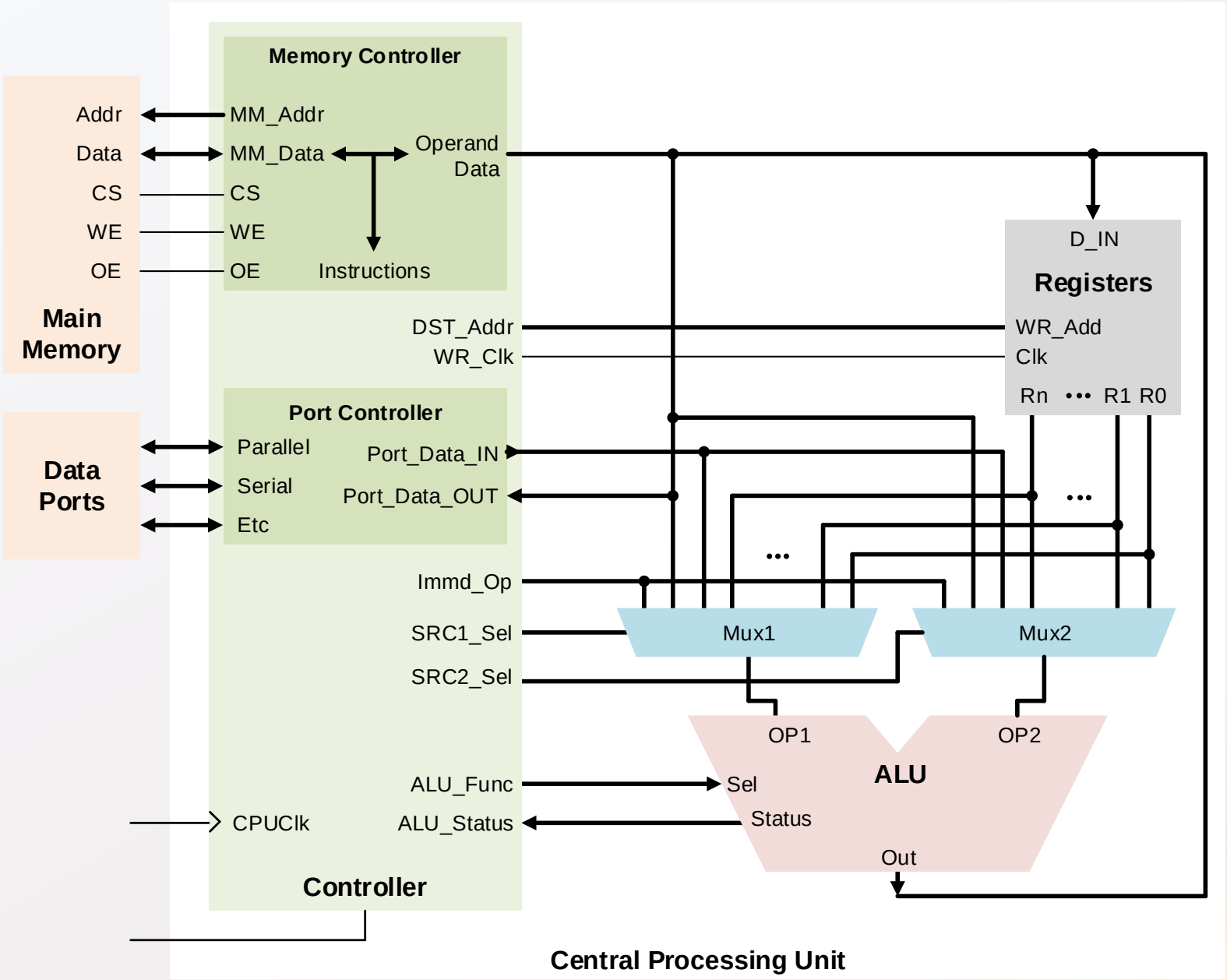
In this diagram, every register output is routed to the mux inputs. So any register can be routed to the ALU immediately, just by changing the SRC1/SRC2 Sel lines.

ALU outputs can also be routed back into a register using a separate address. Is this important?

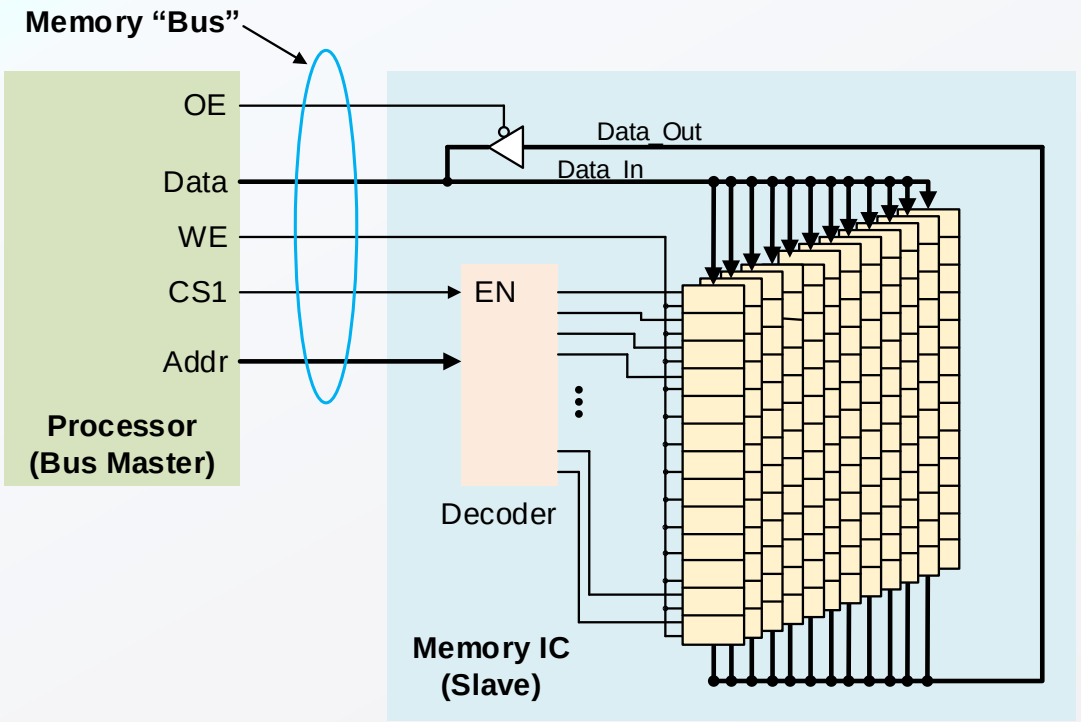
Main Memory

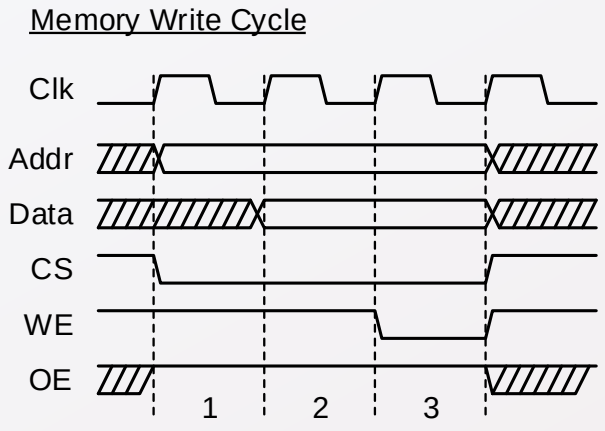
Main memory is typically built from both RAM (Random Access Memory) and ROM (Read Only Memory), and even disc or other media.

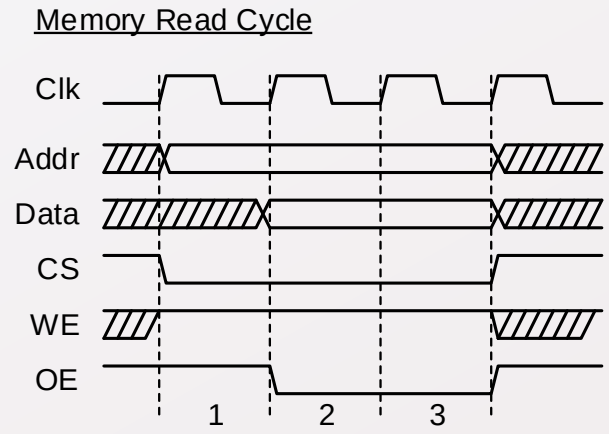
Main Memory stores both program and data, so the external data bus must be shared.



Main Memory





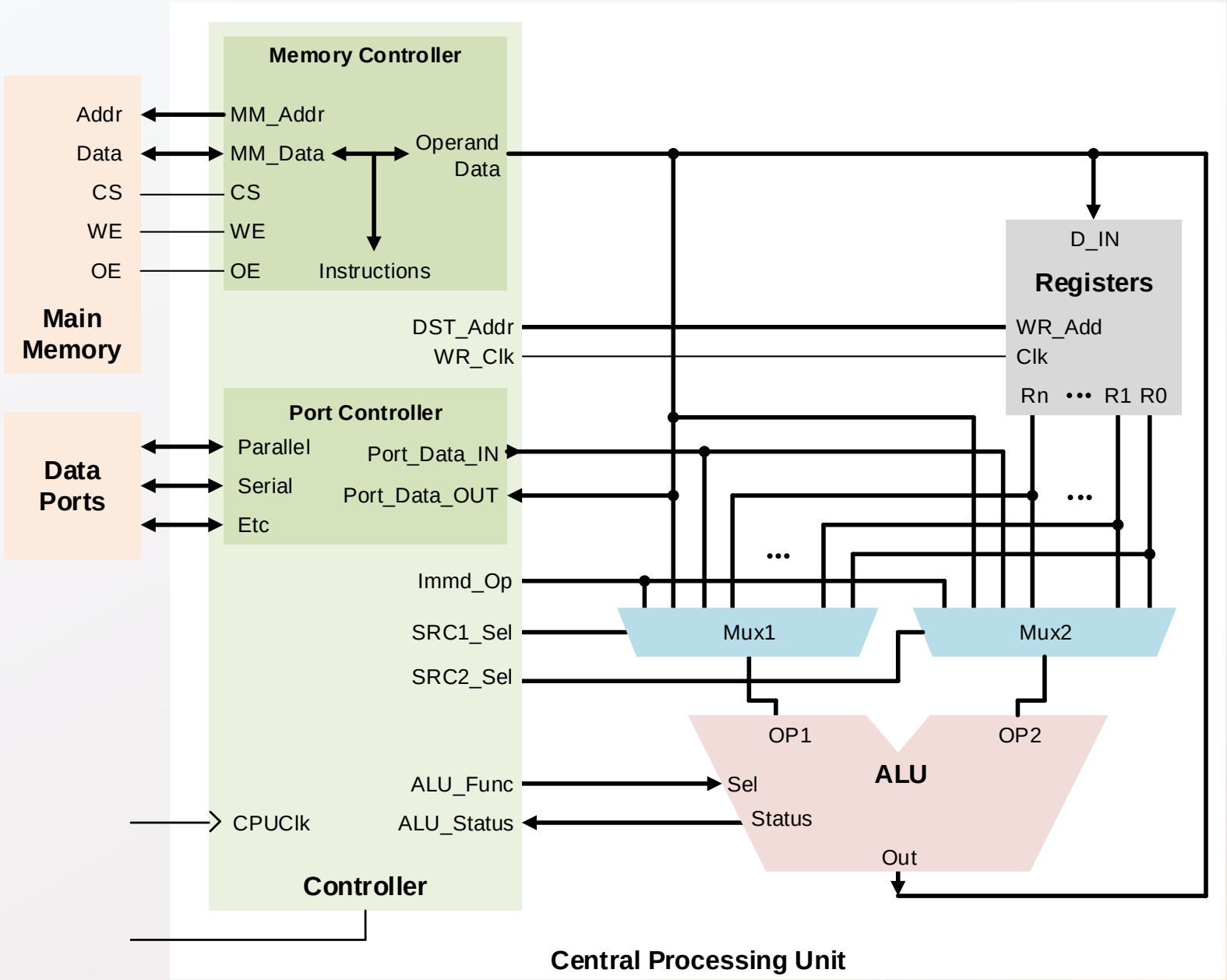


To access main memory, the processor enables the memory device and places an address on the address bus. For a write cycle, the processor places data on the data bus, and then pulses "WE" to cause the data to be latched. For a read cycle, the processor "turns on" the memory device by driving OE, and then latches the data on the next clock cycle. When an instruction is read from main memory, it is called a "fetch".

Controller

The controller is a state machine that operates a basic, repeating “instruction cycle”:

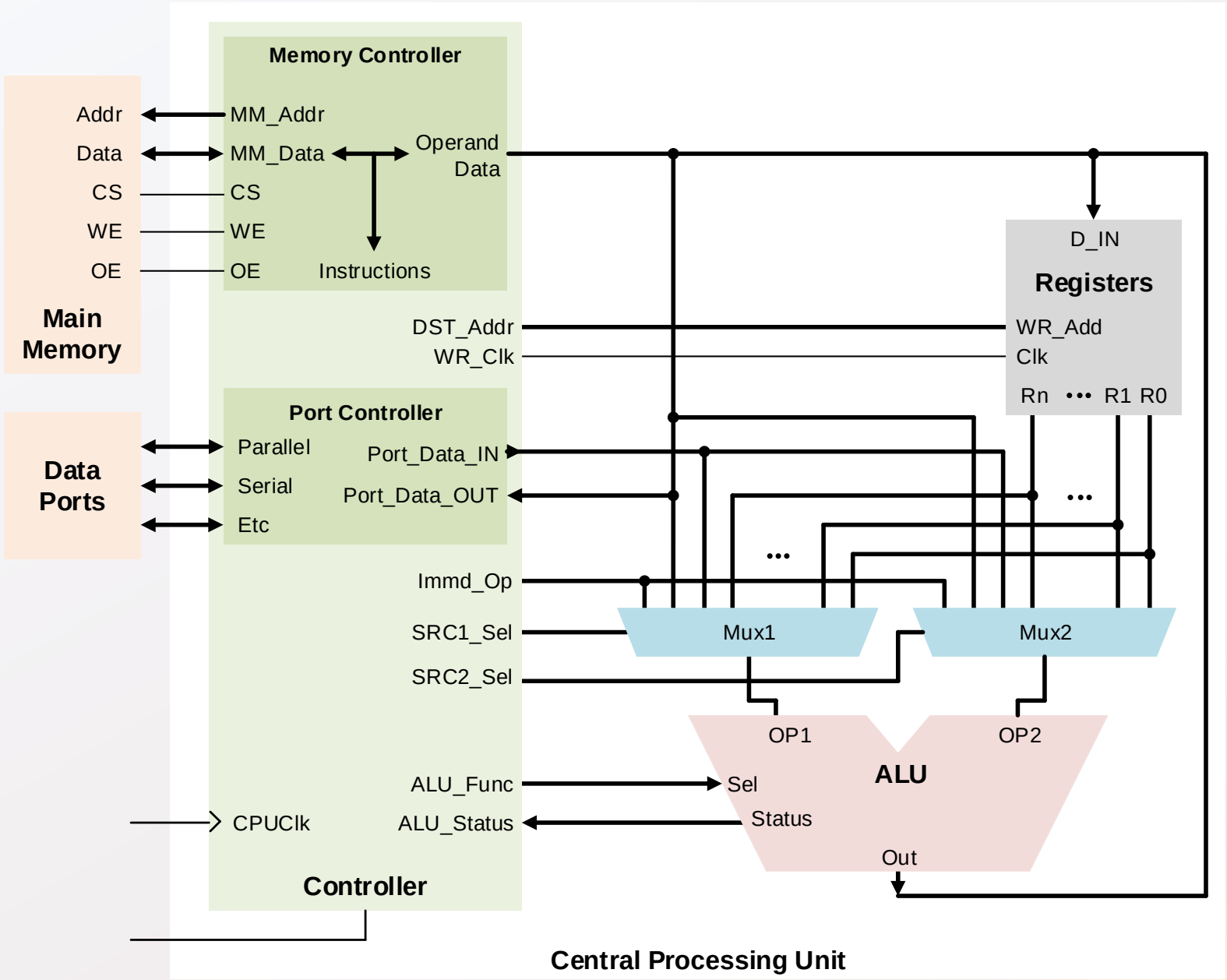
1. “Fetch” opcode from main memory;
2. Decode operand to select the data source, data destination, and ALU function
3. Drive timing signals to “execute” the instruction. Since the muxes and ALU are combinational, timing amounts to triggering data storage elements.



Controller

The clock period is chosen to allow enough time for operands to flow through the muxes, combine in the ALU, and route back to the destination. Typical clocks range from 8MHz – 200MHz.

High-performance processors run as fast as possible for maximum throughput. Embedded processors typically run slower to conserve power.



In our example, lets assume:

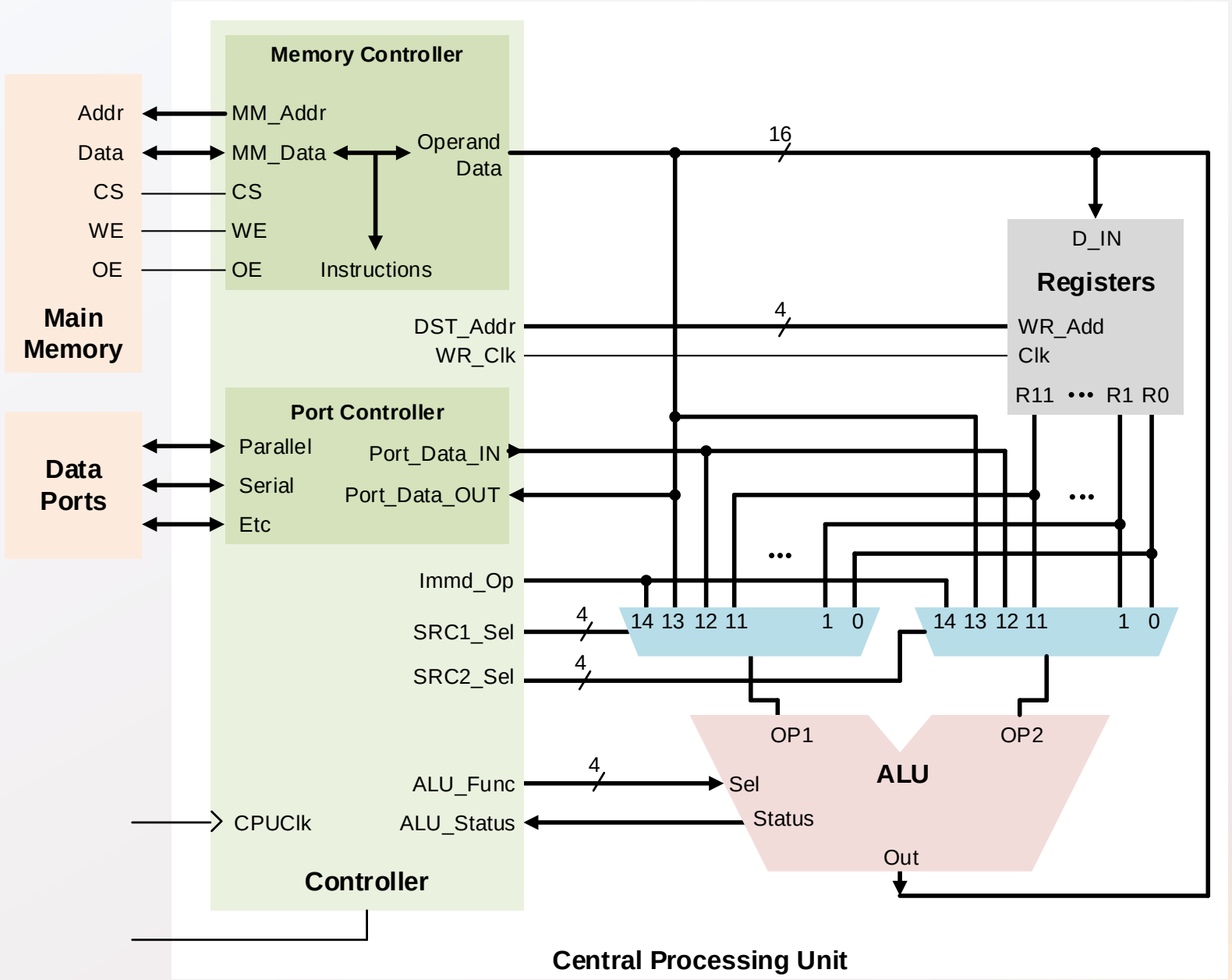
16-bit data busses

12 16-bit Registers

15 Input Bus muxes (4-bit select)

16 Function ALU:

ALU Function	Code	Operation
ADD	0000	Out <= OP1+OP2
INC	0001	Out <= OP1+1
SUB	0010	Out <= OP1-OP2
DEC	0011	Out <= OP1-1
ADDI	0100	Out <= OP1+Immd
SUBI	0101	Out <= OP1-Immd
ASL	0110	Out <= OP1<<Val
ASR	0111	Out <= OP1>>Val
CLR	1000	Out <= 0
NOT	1001	Out <= !OP1
AND	1010	Out <= OP1&OP2
OR	1011	Out <= OP1|OP2
XOR	1100	Out <= OP1^OP2
XFER	1101	Out <= OP1
BZ	1110	Out <= OP1
JMP	1111	Out <= OP1

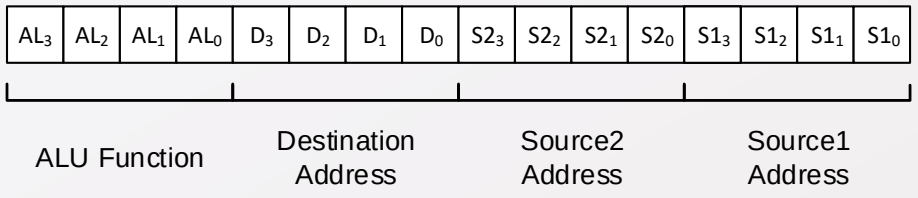


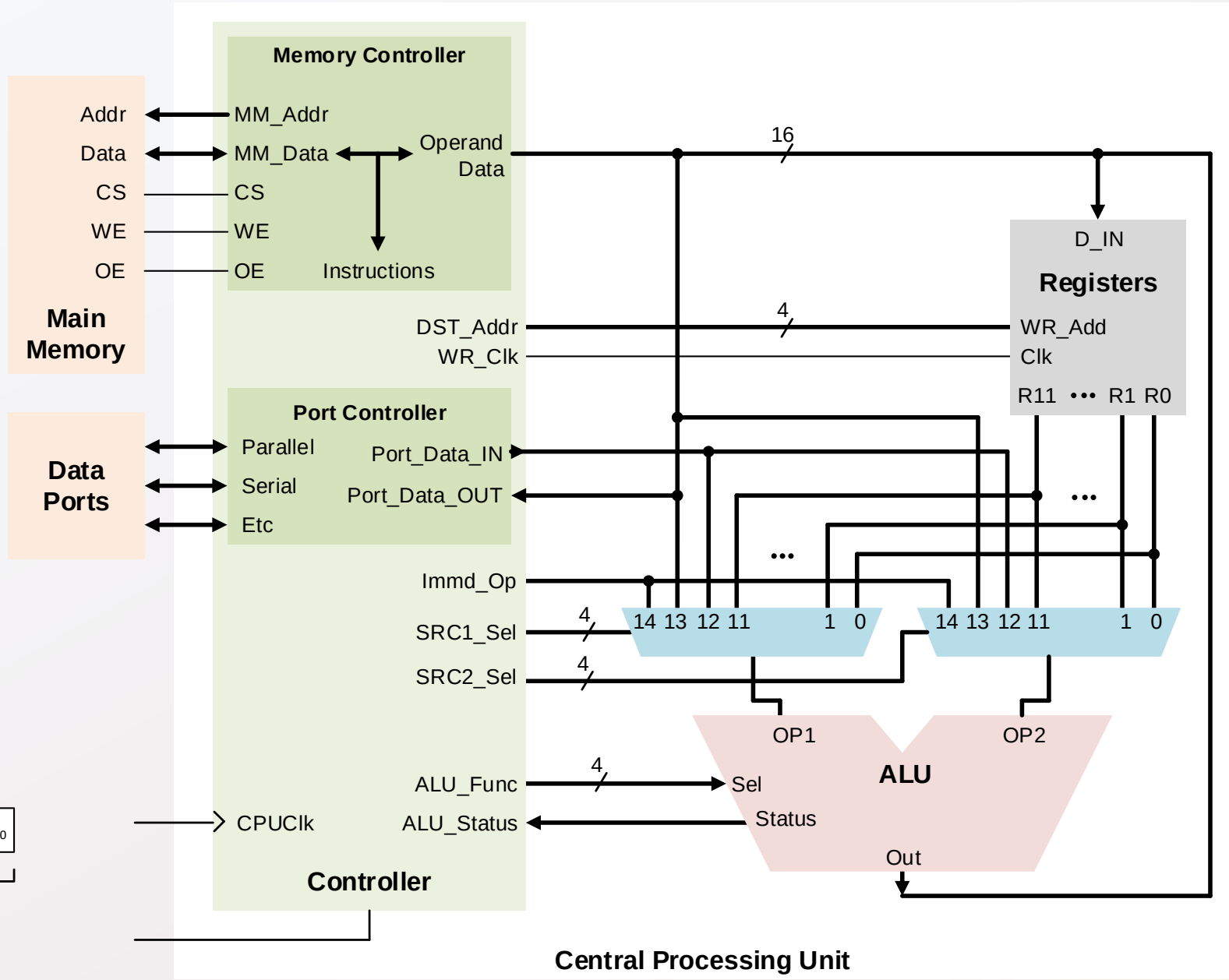
To execute an operation, the controller must:

- Select a source for OP1 (SRC1_Sel);
- Select a source for OP2 (SRC2_Sel);
- Select a destination (DST_Addr);
- Select an ALU function

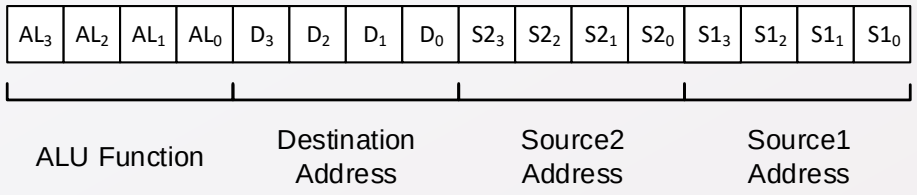
This requires a 16-bit "operation code"

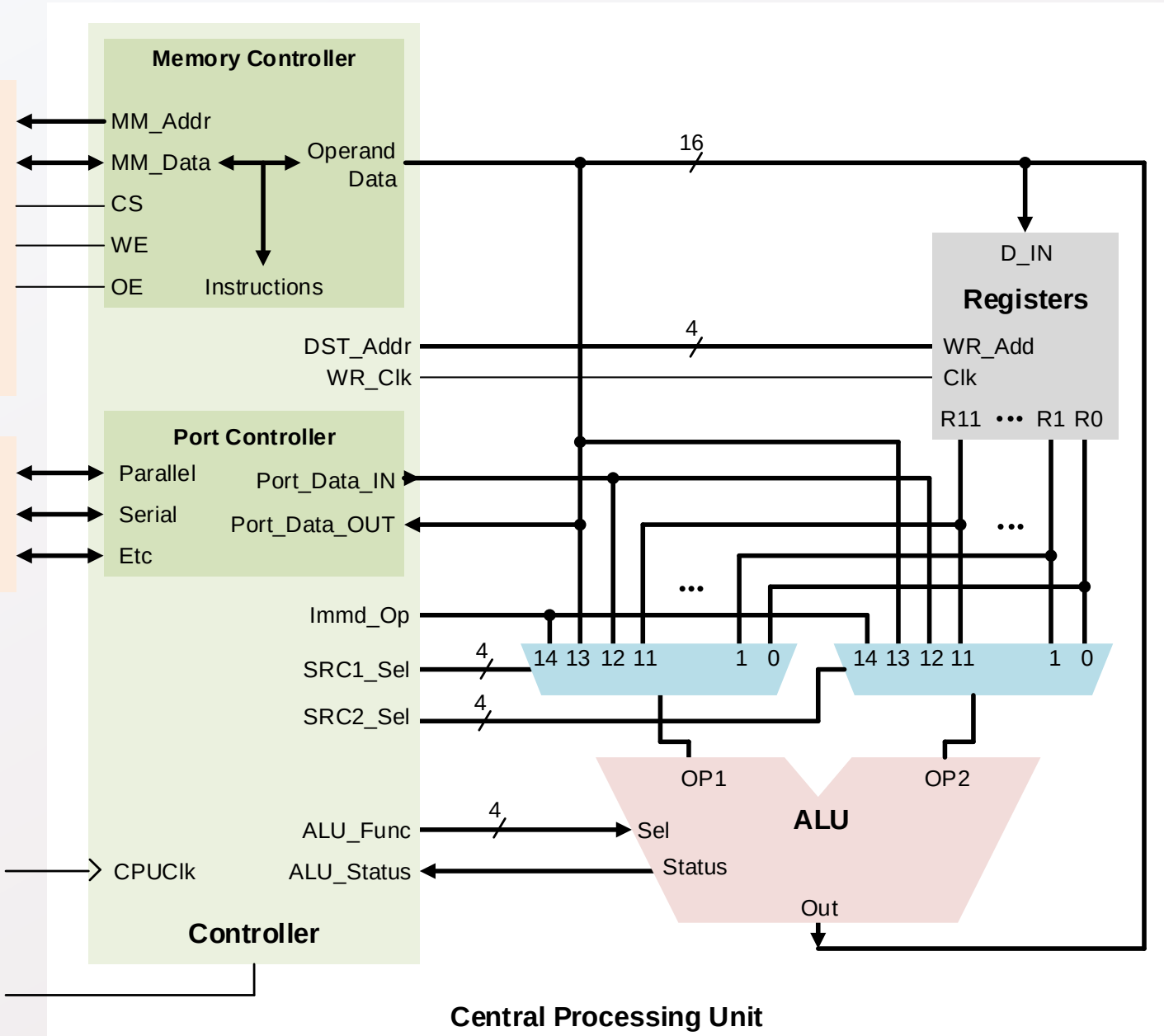
Opcodes typically have certain defined fields of bits to control the ALU and select operands.





ALU Function	Code	Operation
ADD	0000	Out <= OP1+OP2
INC	0001	Out <= OP1+1
SUB	0010	Out <= OP1-OP2
DEC	0011	Out <= OP1-1
ADDI	0100	Out <= OP1+Immd
SUBI	0101	Out <= OP1-Immd
ASL	0110	Out <= OP1<<Val
ASR	0111	Out <= OP1>>Val
CLR	1000	Out <= 0
NOT	1001	Out <= !OP1
AND	1010	Out <= OP1&OP2
OR	1011	Out <= OP1|OP2
XOR	1100	Out <= OP1^OP2
XFER	1101	Out <= OP1
BZ	1110	Out <= OP1
JMP	1111	Out <= OP1



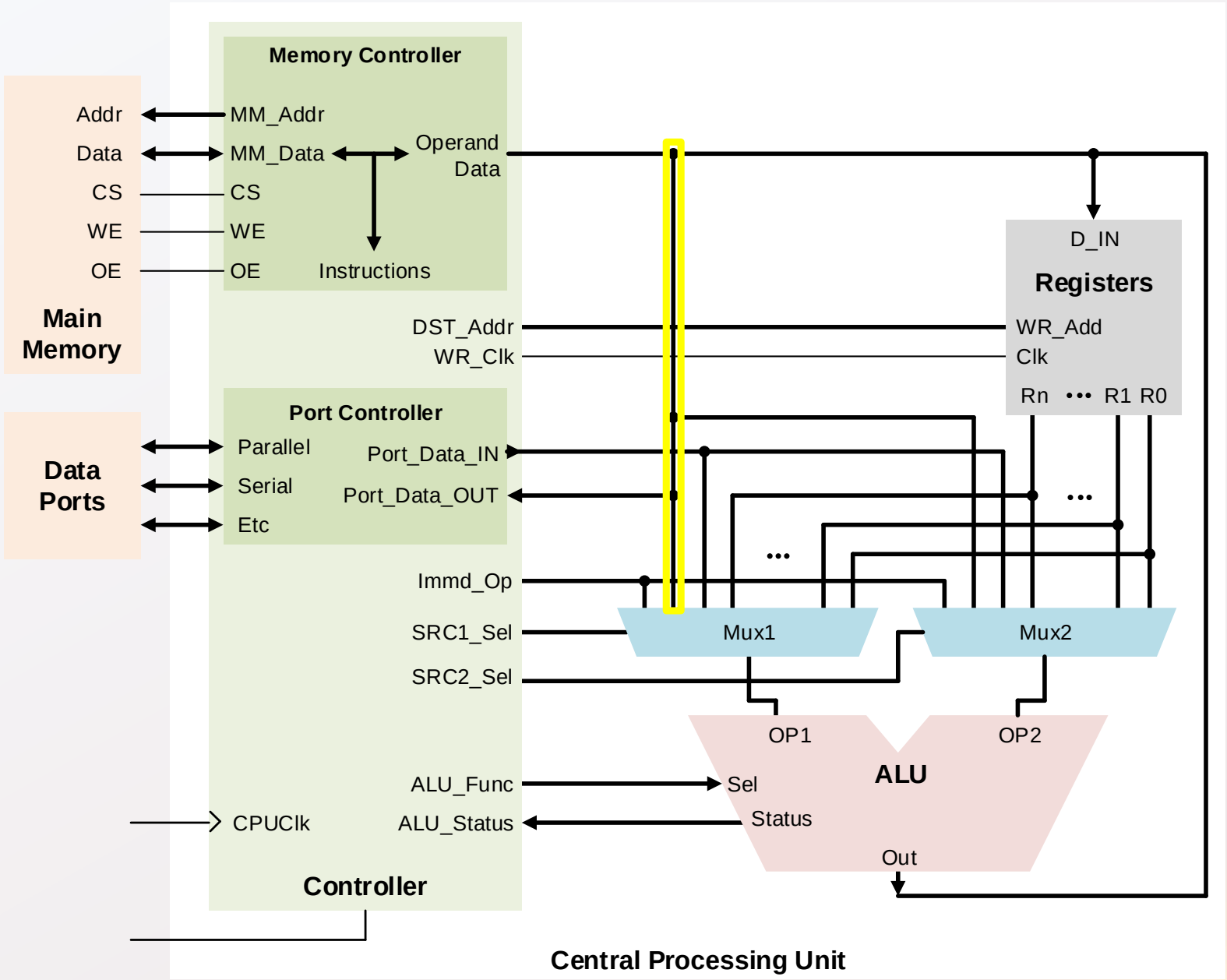


Opcode 0000001100100001 adds the contents of R2 and R1, and stores the result in R3 at the next edge or WR_Clk

In our diagram, one source input can come from main memory (mux input 13).

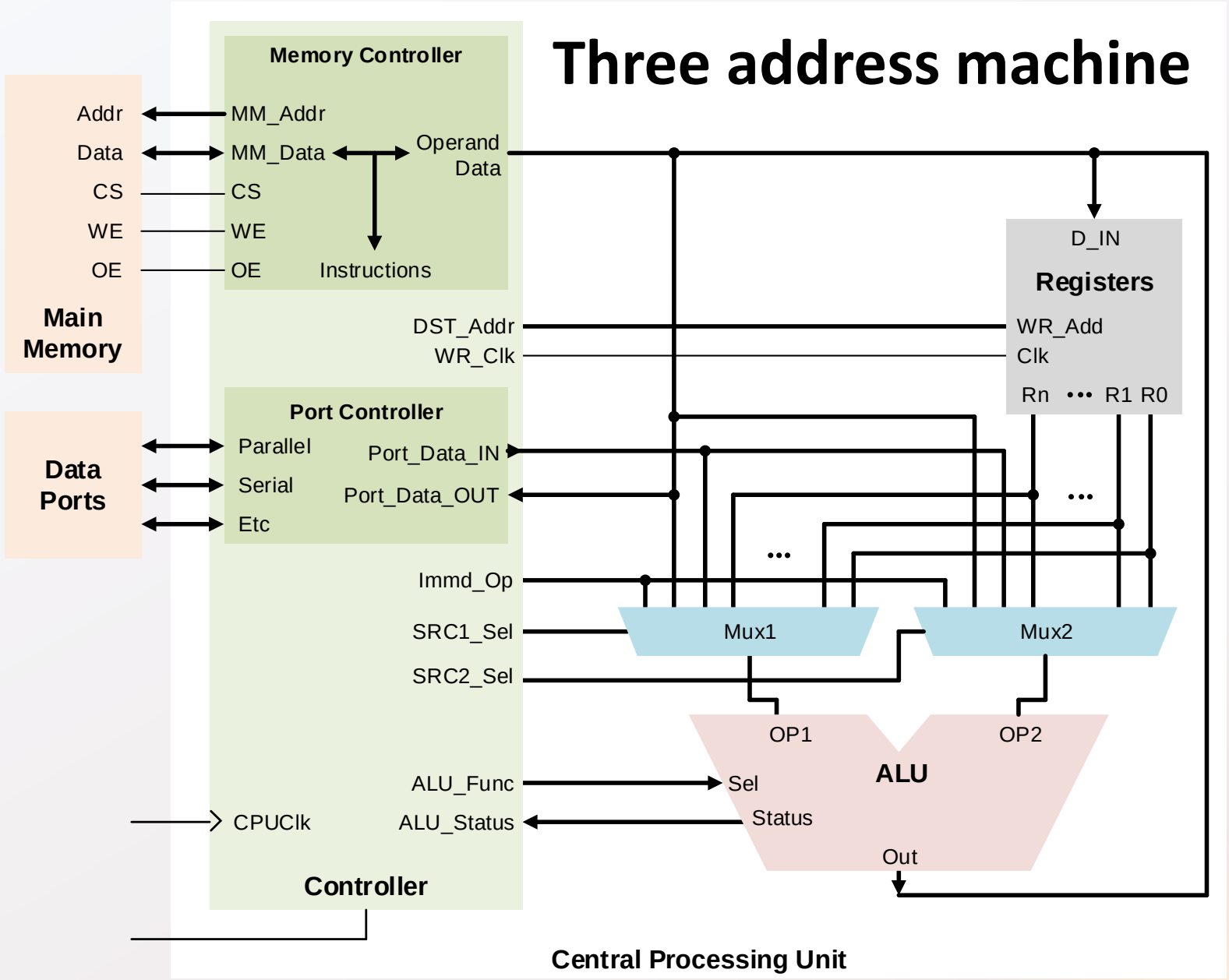
In many processors, ALU operands can only come from registers, and not from main memory.

Processors that only allow ALU operands to come from registers are called “load-store” architectures, because all ALU operations must be preceeded with a load, and all results must be stored in main memory. This is a common feature found in “RISC” processors.



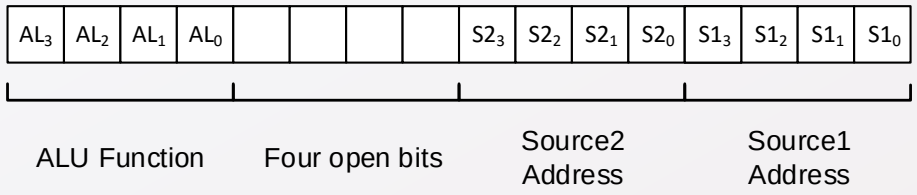
This CPU is a “three-address machine”, because the controller generates unique addresses for both source operands and for the destination. Three-address processors are fast and efficient, but they pay a price…

Some CPUs use two, or even one address for both source operands and the destination. Can you think of a reason why?

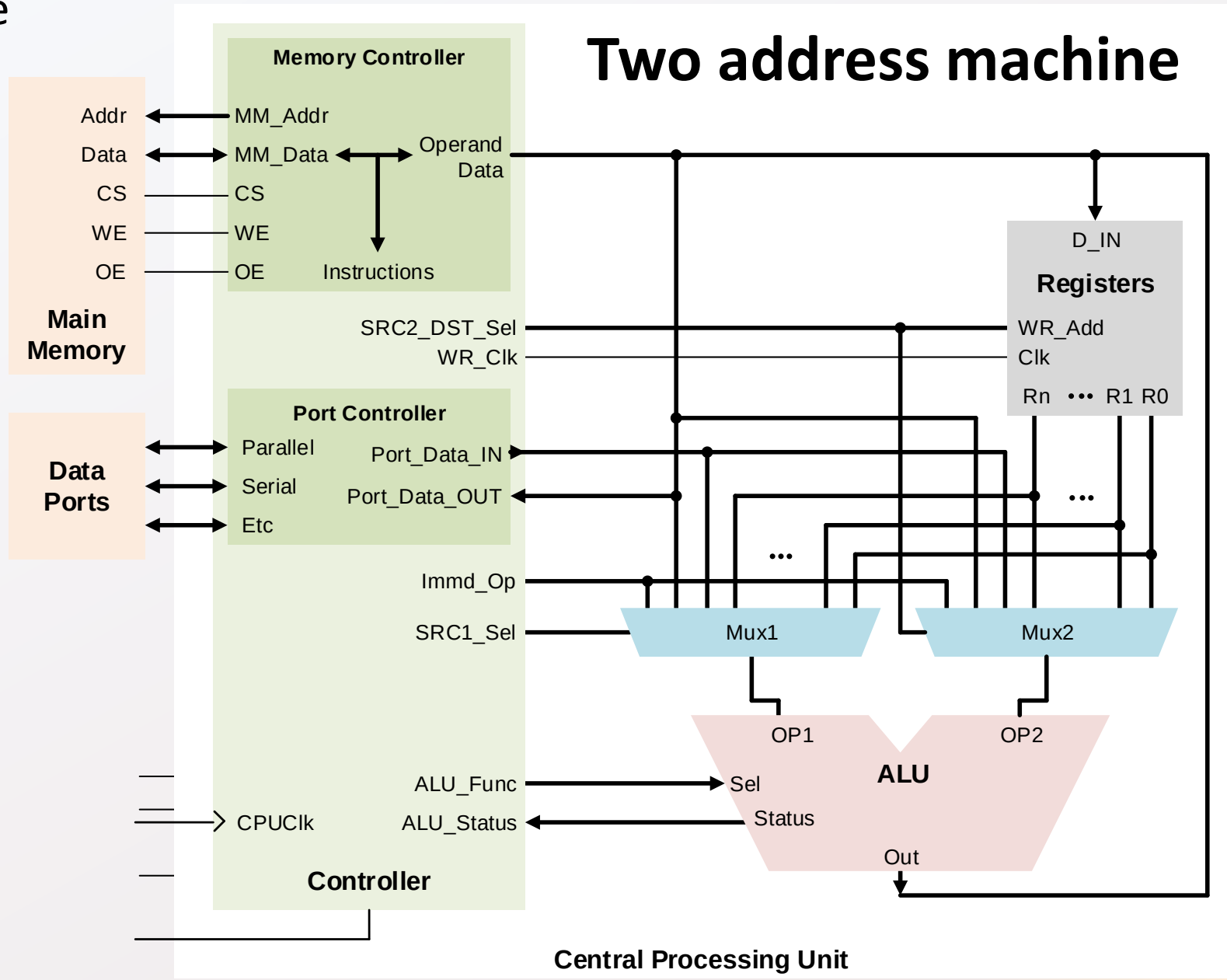


In a 2-address machines, one source operand must also be the destination.

This saves bits in the opcode, and more opcode bits means more options for the ALU and controller. (In our 16-bit example, we only had room for 16 ALU commands – pretty sparse!)

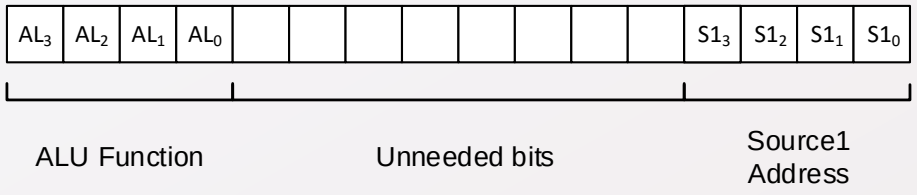


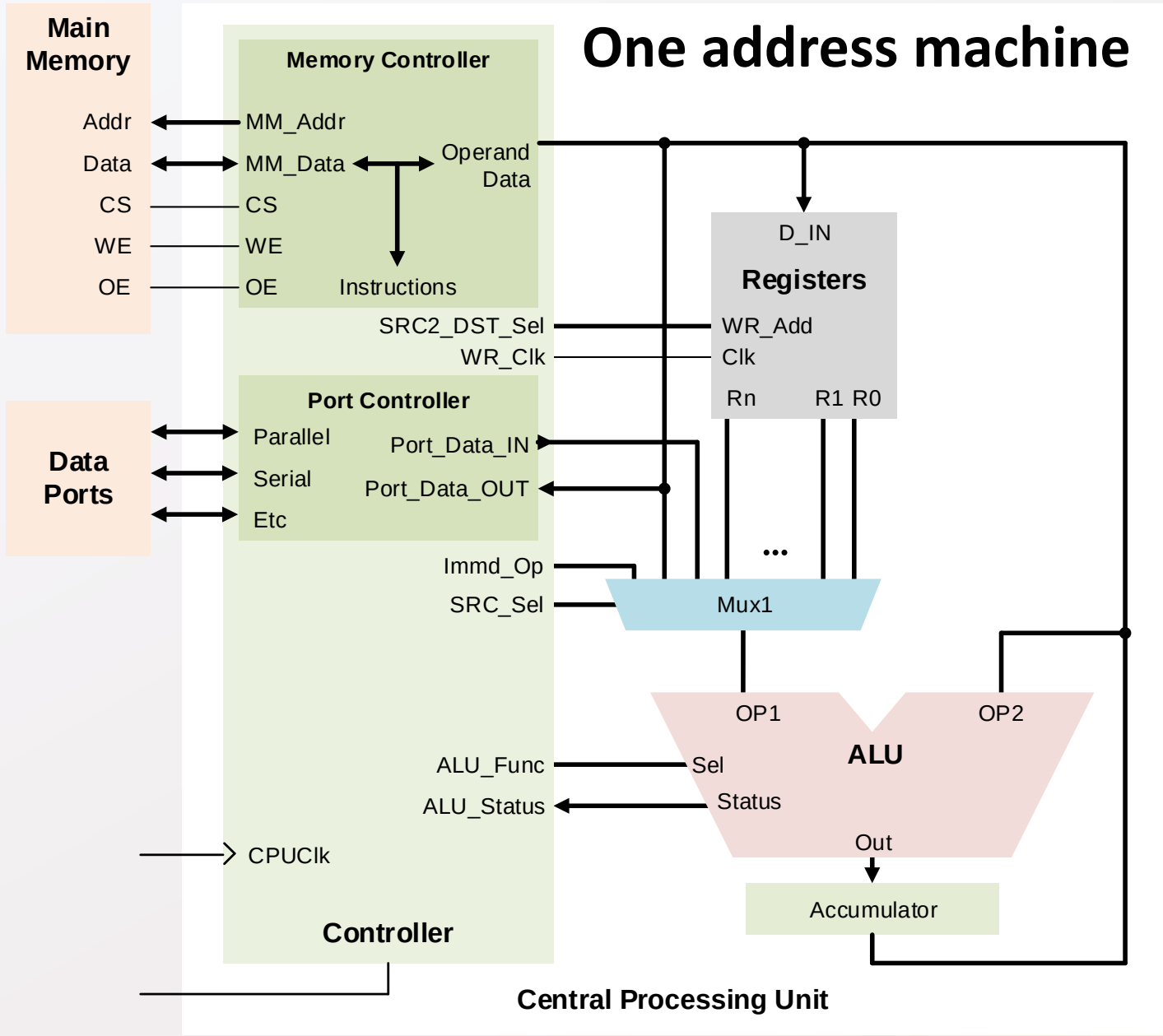
Two address machine



One-address machines require that all ALU results are held in an accumulator, and the accumulator is always one of the operands. So, only one source needs to be specified.

This was a popular architecture for 8-bit processors.



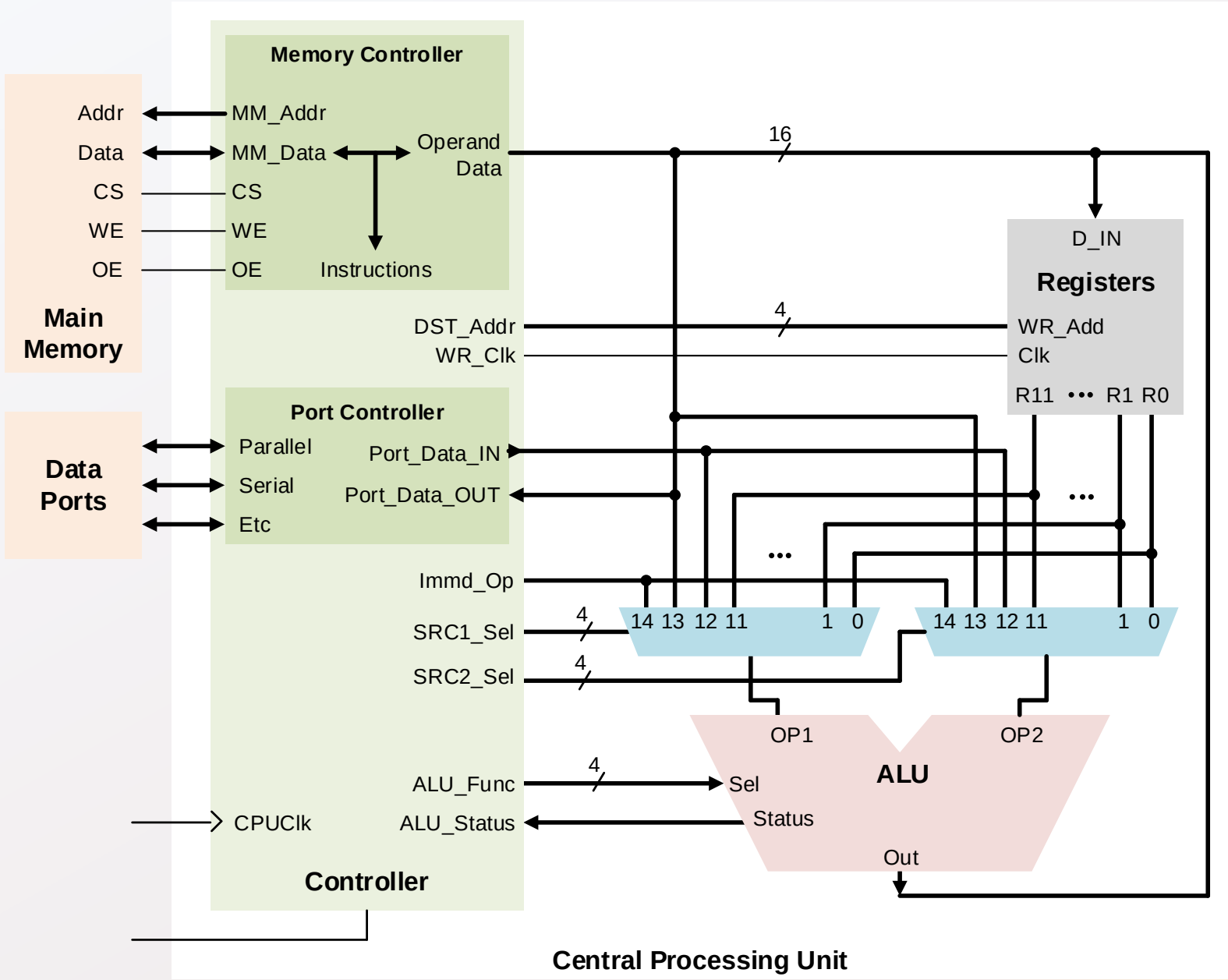


Can you identify an opcode to:

Decrement the contents of R1, and store the result in R5?

Invert the contents of R2, and store the result back in R2? (Will this work?)

ALU Function	Code	Operation
ADD	0000	Out <= OP1+OP2
INC	0001	Out <= OP1+1
SUB	0010	Out <= OP1-OP2
DEC	0011	Out <= OP1-1
ADDI	0100	Out <= OP1+Immd
SUBI	0101	Out <= OP1-Immd
ASL	0110	Out <= OP1<<Val
ASR	0111	Out <= OP1>>Val
CLR	1000	Out <= 0
NOT	1001	Out <= !OP1
AND	1010	Out <= OP1&OP2
OR	1011	Out <= OP1|OP2
XOR	1100	Out <= OP1^OP2
XFER	1101	Out <= OP1
BZ	1110	Out <= OP1
JMP	1111	Out <= OP1



Major Points

Understand blocks

- ALU and Muxes
- Registers
- Main Memory
- Controller & Instruction Cycle

Understand Opcodes

- Control ALU and operand address
- Trade-offs (more bits)

General Models