Revised: 3/24/2021

Processor Architecture I

Michael C. Hackett

Assistant Professor, Computer Science

Community College of Philadelphia

Lecture Topics

- Computer Architecture
- Arithmetic Logic Unit
- Storage Components
- Other Components
- Control Unit
- Processor Performance
- Amdahl's Law

Computer Architecture

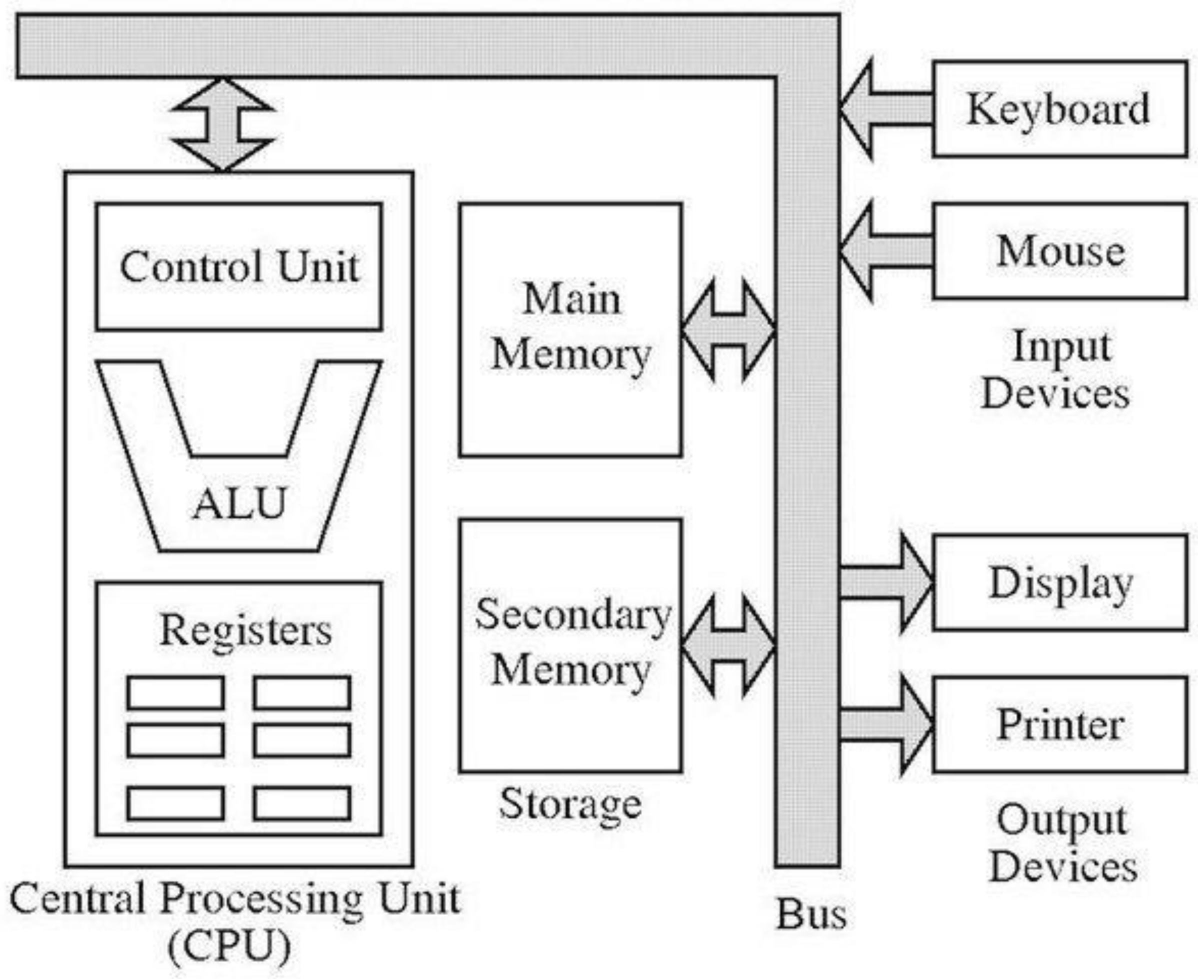
- *Computer organization* (as you've gathered by now) refers to how the functional components of a computer system work
  - Computer organization also goes by the name "microarchitecture"
- *Computer architecture* refers to the design of the systems that these functional components are a part of.
  - Normally following a freshman/sophomore-level computer organization course is a junior/senior-level course in computer architecture
- The remainder of this course will be a high-level study of computer/processor architecture.
  - With some low-level looks at points of interest and importance.

Computer Architecture

- The **von Neumann Architecture** is the computer architecture that modern processors are based on.
  - Named for a computer architecture originally designed by mathematician John von Neumann in 1945

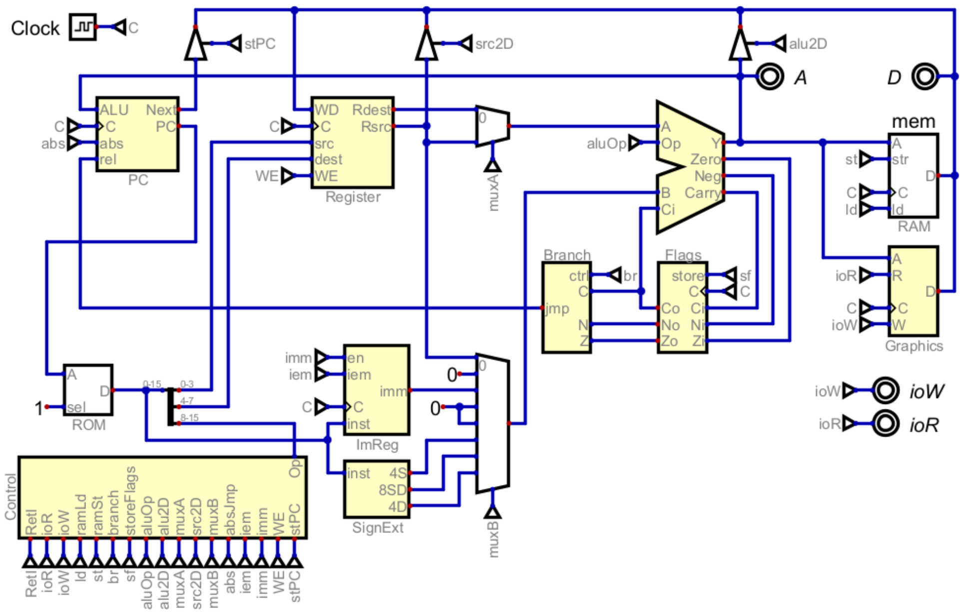
- The von Neumann architecture consists of:
  - The Central Processing Unit (CPU)
    - Arithmetic Logic Unit (ALU)
    - Register file
    - Control unit
  - Memory for storing data and instructions
  - External storage
  - Input and output devices

The von Neumann Architecture

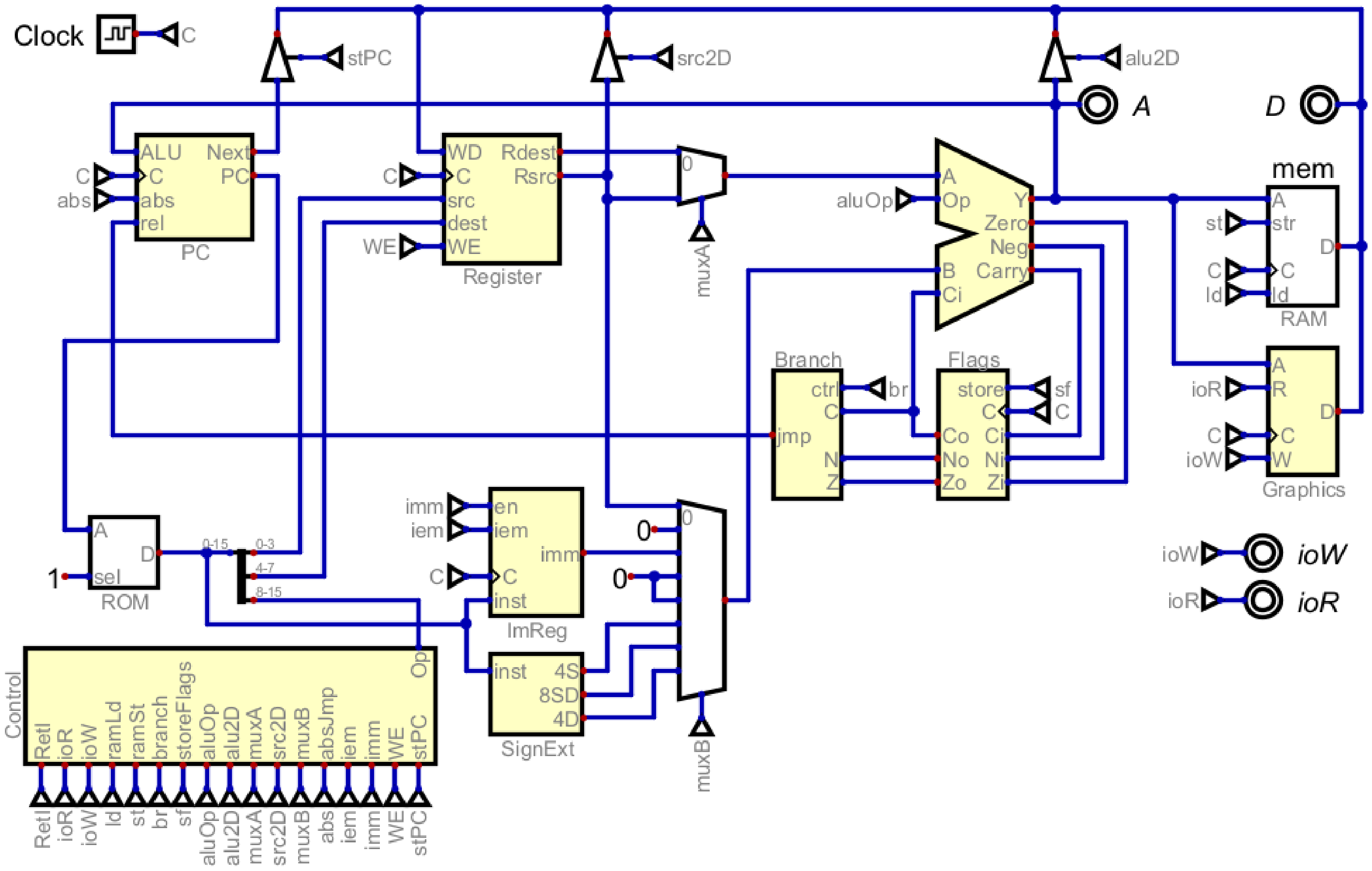


Datapath

- A **datapath** refers to the components, storage elements, and connections in the processor’s architecture.
  - Illustrated in this block diagram of a 16-bit processor (larger on next slide):



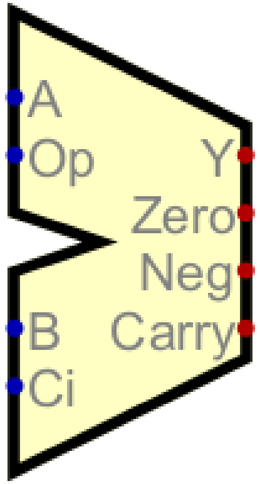
Datapath

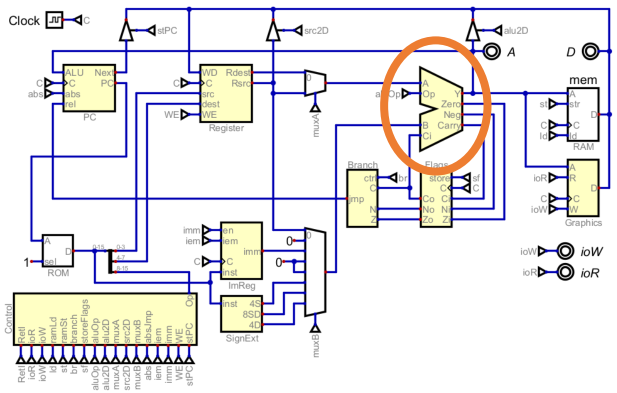

Clock
stPC
src2D
alu2D
A
D
PC
ALU
Next PC
abs
rel
Register
WD
Rdest
Rsrc
src
dest
WE
muxA
aluOp
Zero
Neg
Carry
mem
RAM
str
ld
Graphics
ioR
ioW
Branch
ctrl
jmp
Flags
store
sf
br
ROM
sel
ImReg
en
iem
imm
inst
SignExt
4S
8SD
4D
muxB
Control
RetI
ioR
ioW
ramLd
ramSt
branch
storeFlags
aluOp
alu2D
muxA
src2D
muxB
absJmp
iem
imm
WE
stPC
Op

Arithmetic Logic Unit

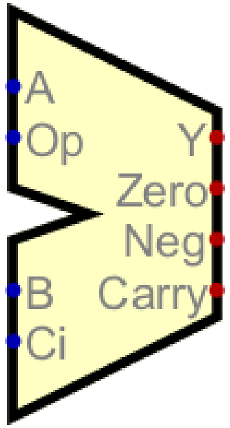
- The ALU performs arithmetic (e.g., addition and subtraction) and logical operations.

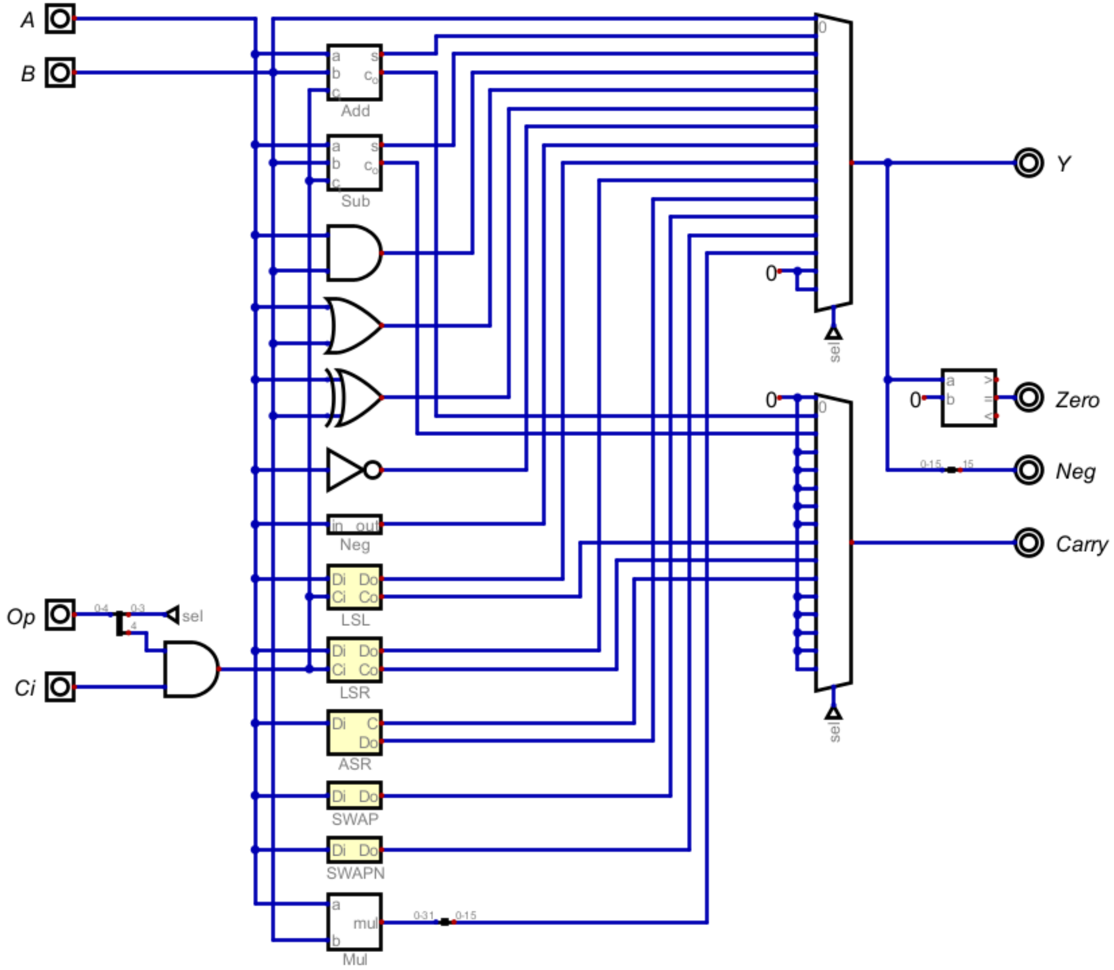
- An ALU typically has:
  - Two data inputs, **A** and **B**
    - The operand(s) to add, subtract, or, xor, etc.
  - **Operation select input**
    - Selects what operation to perform
  - One data output, **Y**
    - The result of the operation
  - **Z** output (1 if result is zero)
- Additional inputs and outputs:
  - Carry **in**
  - **Neg** output (1 if result is negative)
  - **Carry** (carry out)





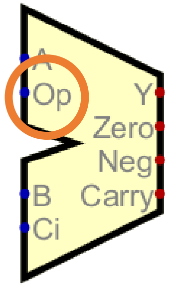
Arithmetic Logic Unit

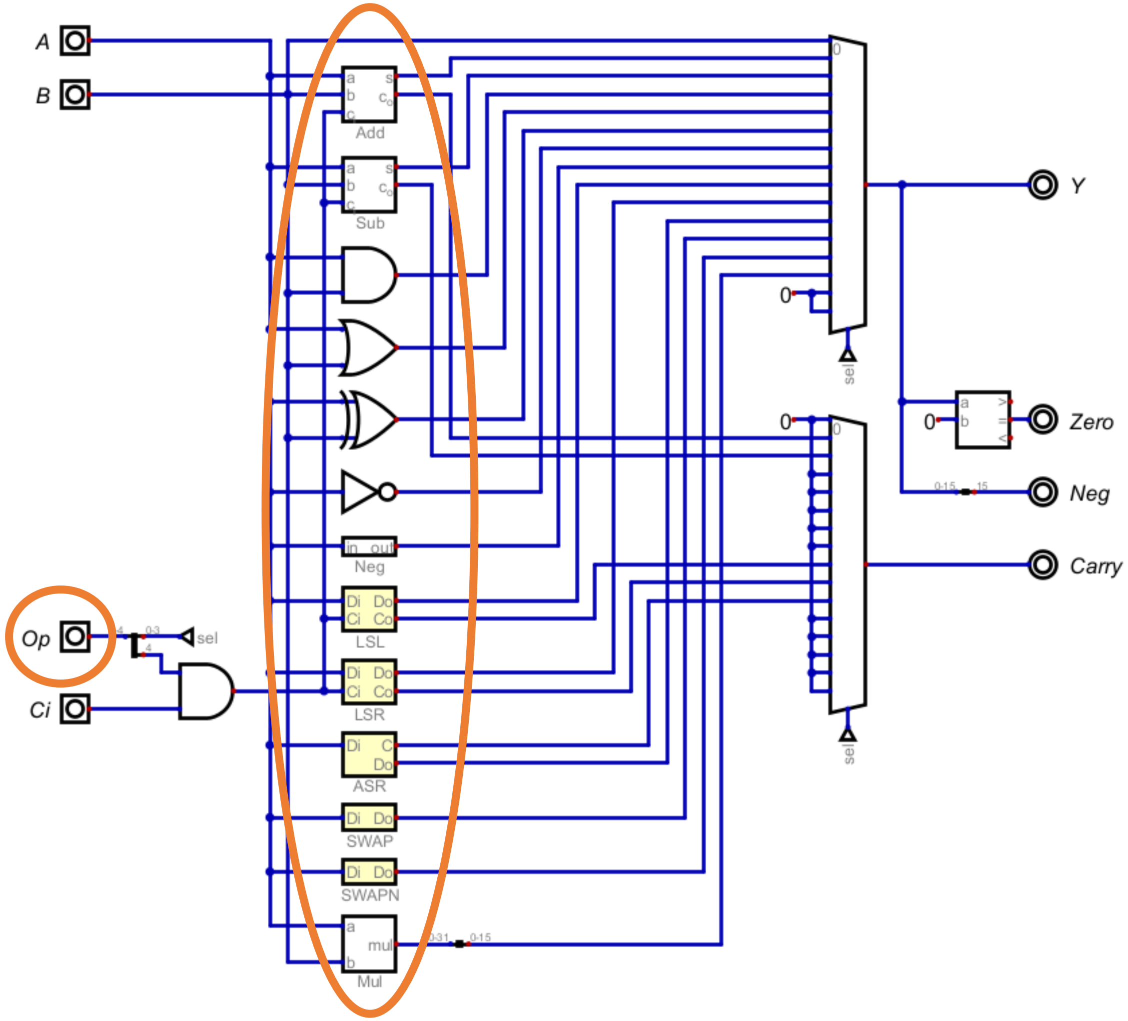




Arithmetic Logic Unit

- Operation selected by Op:
  - 00001 = Add
  - 00010 = Sub
  - 00011 = And
  - 00100 = Or
  - and so on

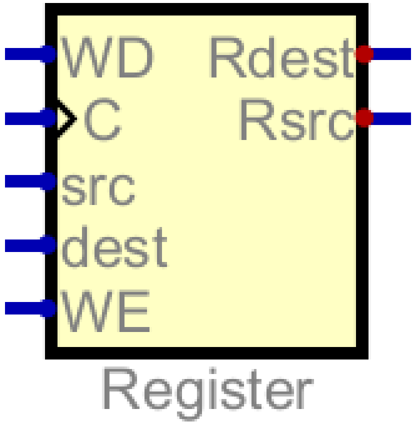


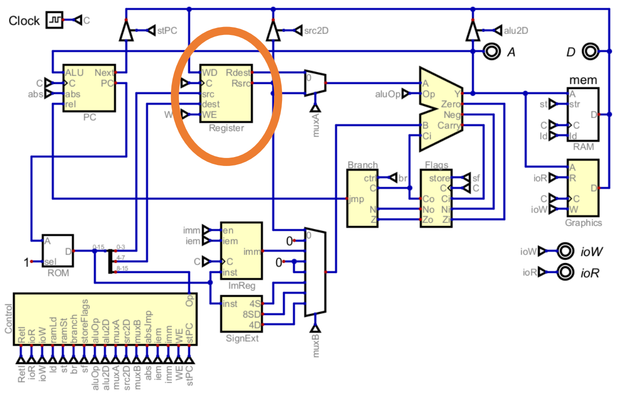


Storage Components

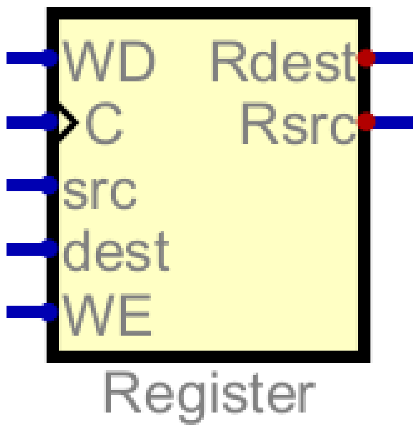
- The **register file** contains the processor’s general purpose registers.

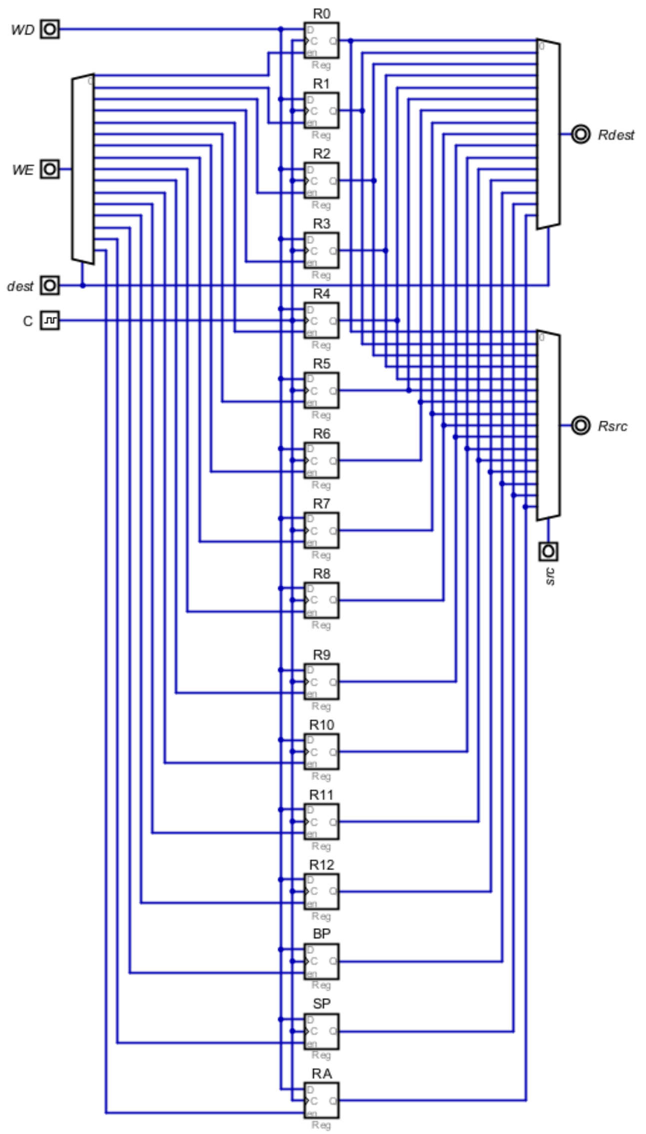
- WD: Data to be stored
- C: Clock
- src: Number of source register
- dest: Number of destination register
- WE: Write enable
- Rdest: Contents of the destination register
- Rsrc: Contents of the source register





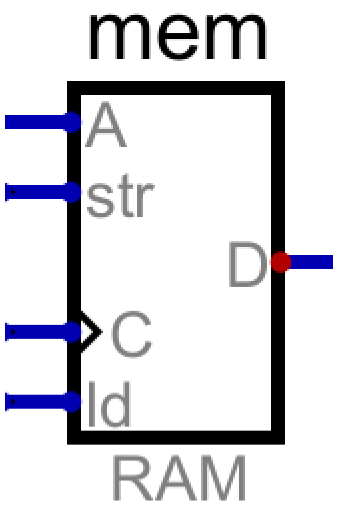
Storage Components



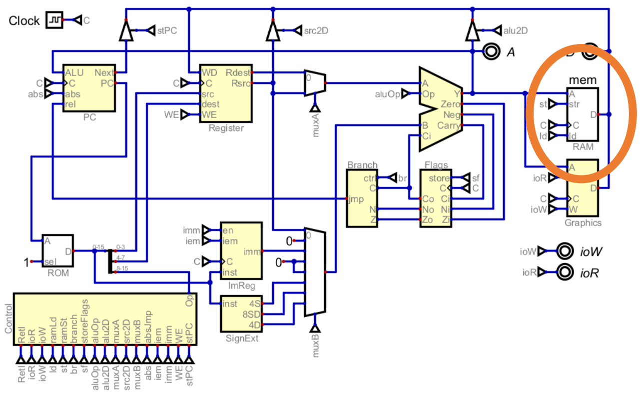


Storage Components

- The **data memory** contains data that can be operands for load and store instructions.



- A: Address
- C: Clock
- str: If 1, store data D
- ld: If 1, load data D

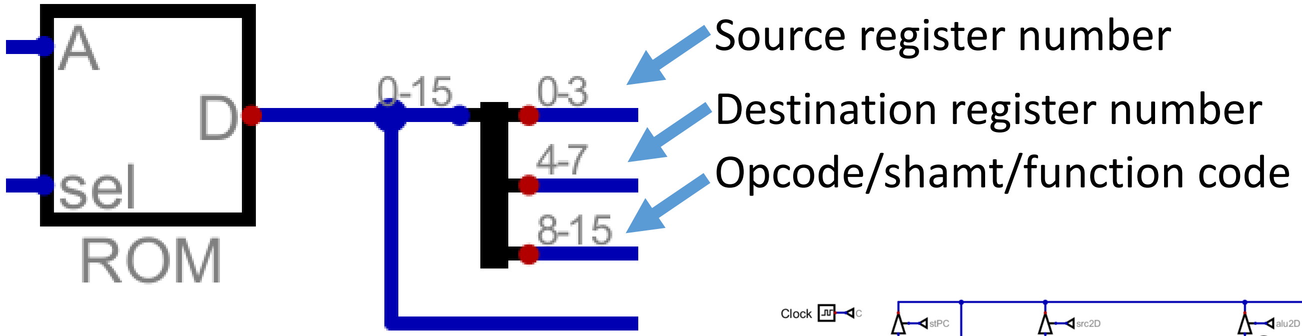


- This example uses a bidirectional pin (D) for reading and writing data

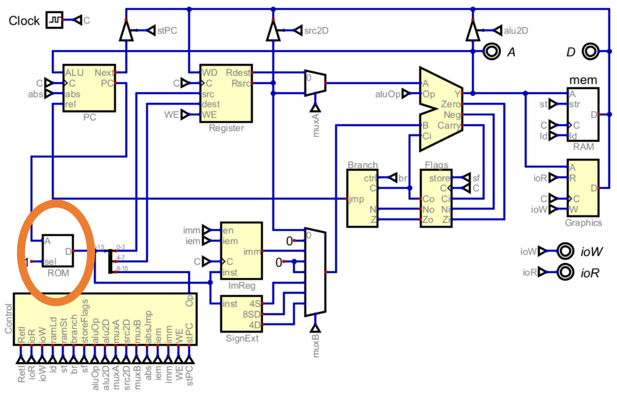
Storage Components

- The **instruction memory** contains program instructions.

- A: Address
- sel: Enables output
- D: Selected data word

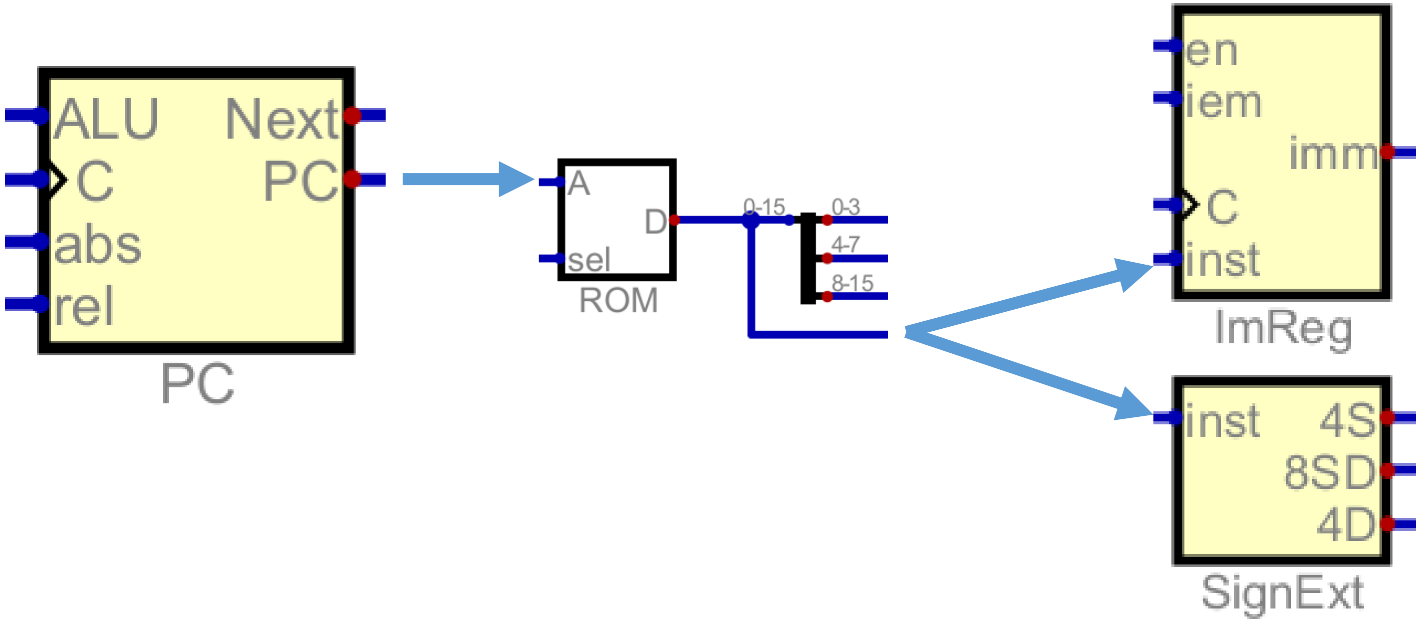


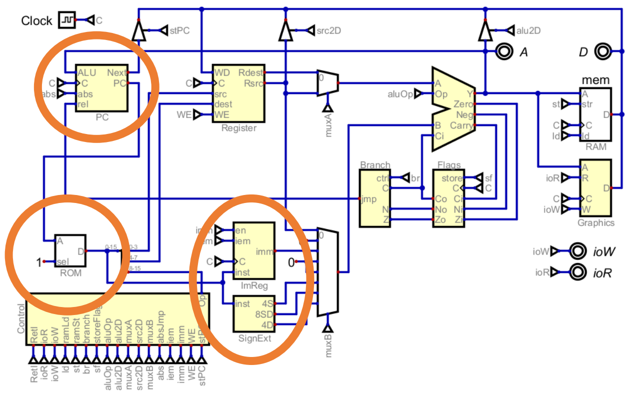
- Implemented as ROM in this example processor



Storage Components

- The instruction memory is always between the **program counter** and **instruction register**.

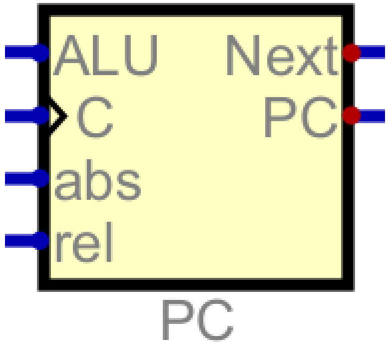


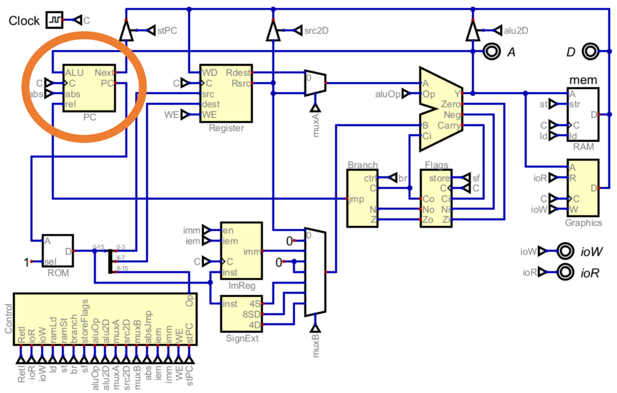


Storage Components

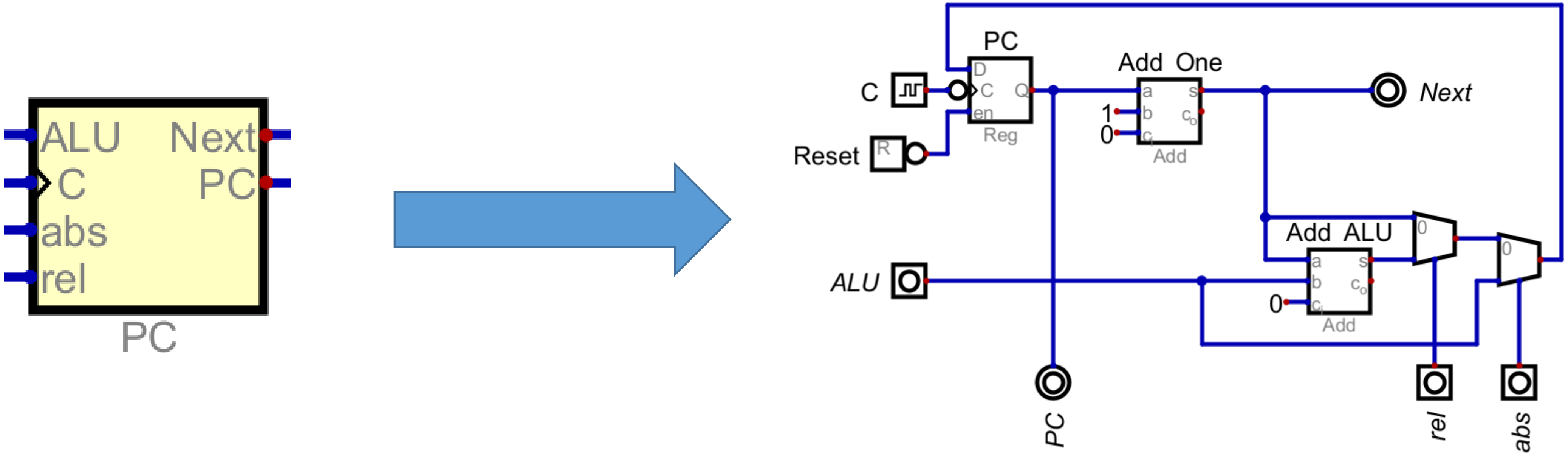
- The **program counter** stored the address of the next instruction to be executed.
  - Every time the CPU executes an instruction, the program counter increments to the next address

- ALU: ALU Output
- C: Clock
- abs: Absolute jump
- rel: Relative jump
- Next: Address of the next instruction
- PC: Address of the current instruction



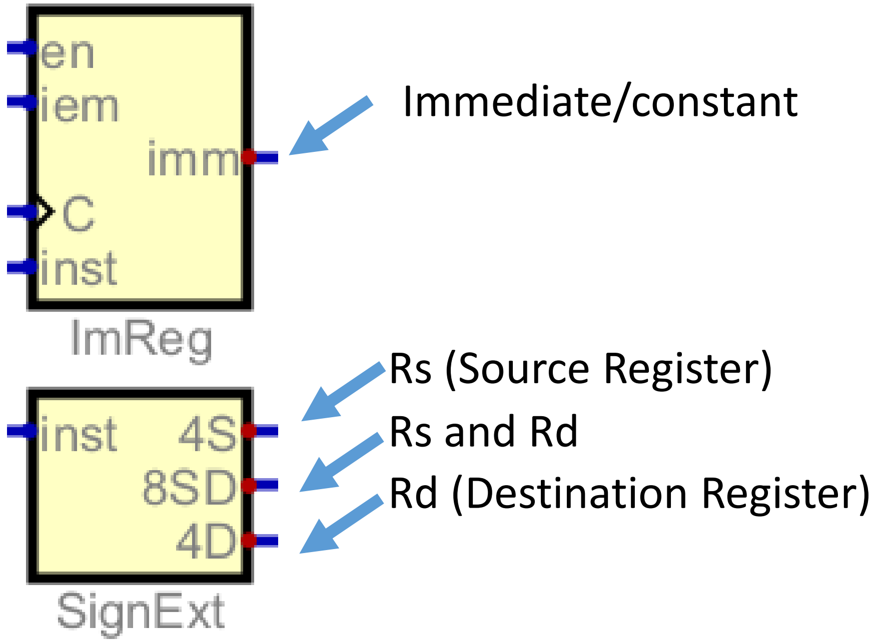


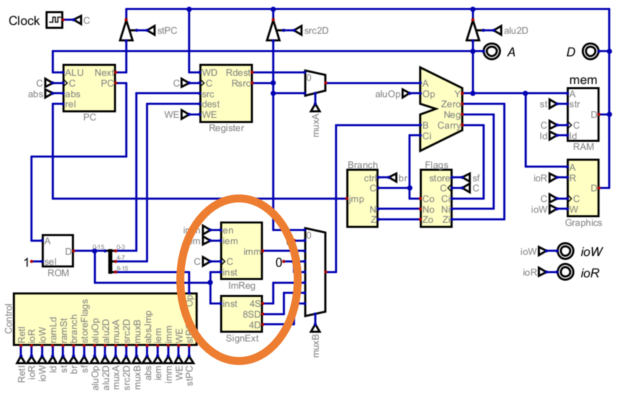
Storage Components



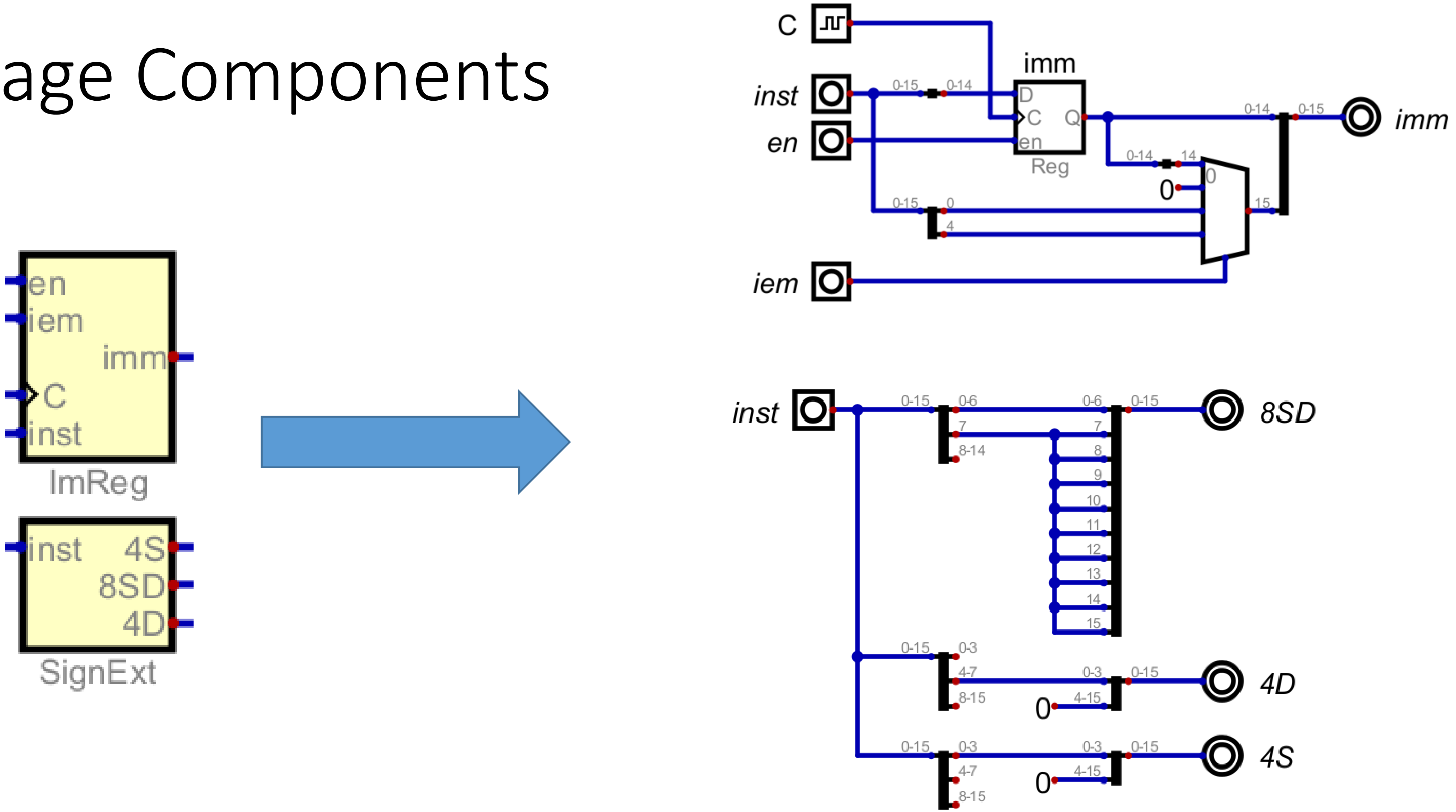
Storage Components

- The **instruction register** stores the instruction currently being executed.
  - It is paired with a sign extend





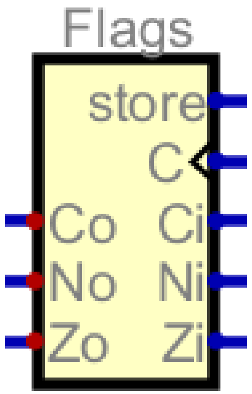
Storage Components

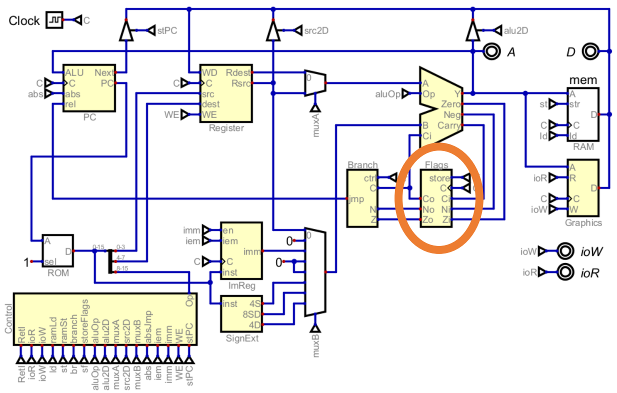


Other Components

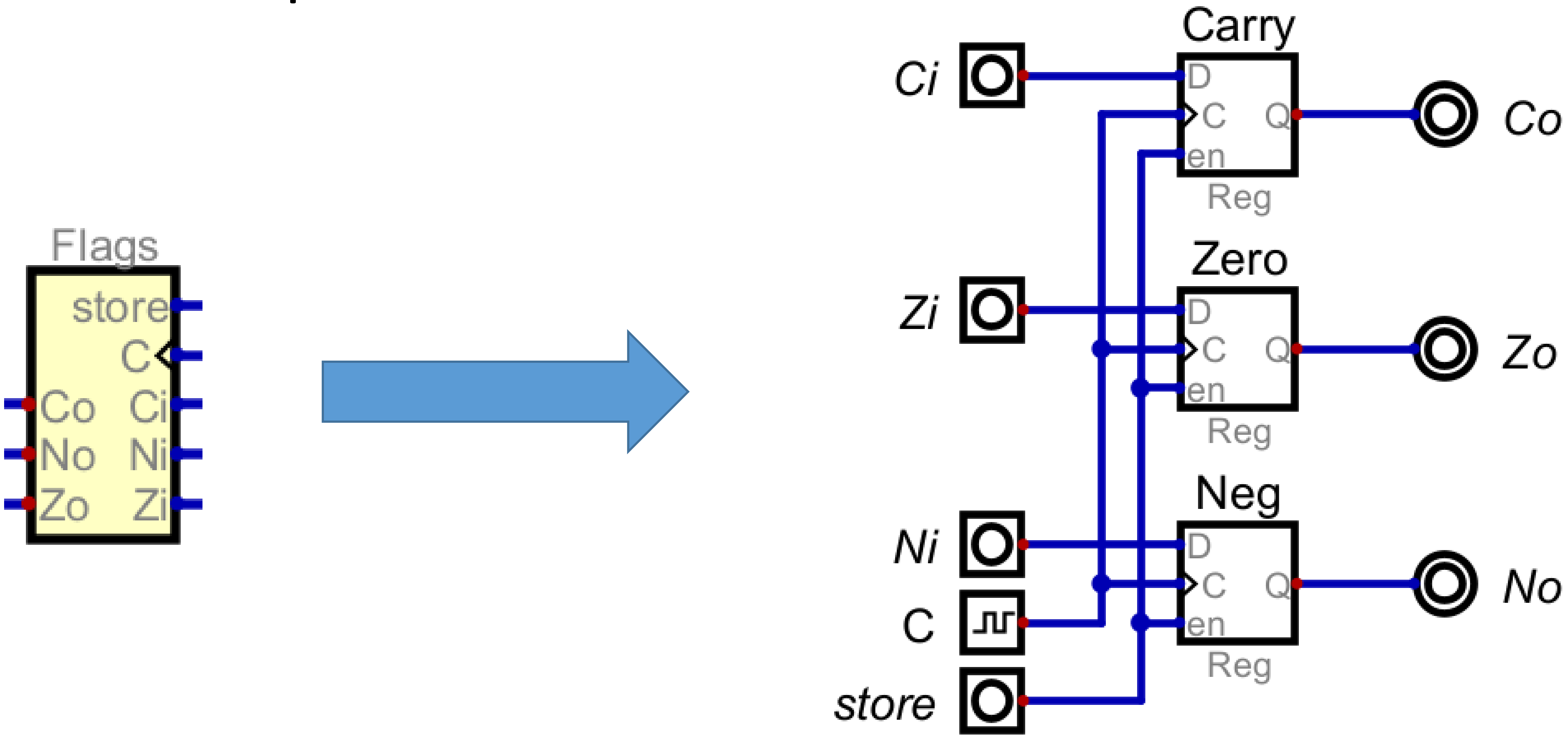
- This processor keeps track of **flags** for handline certain conditions.

- store: Activates storing a flag
- C: Clock
- Ci: Carry in
- Ni: Negative in
- Zi: Zero in
- Co: Carry out
- No: Negative out
- Zo: Zero out





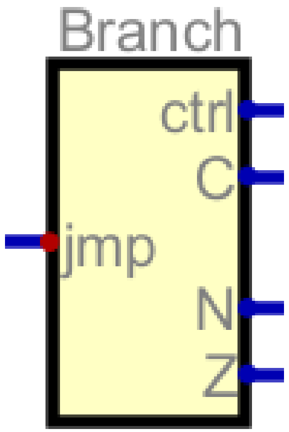
Other Components

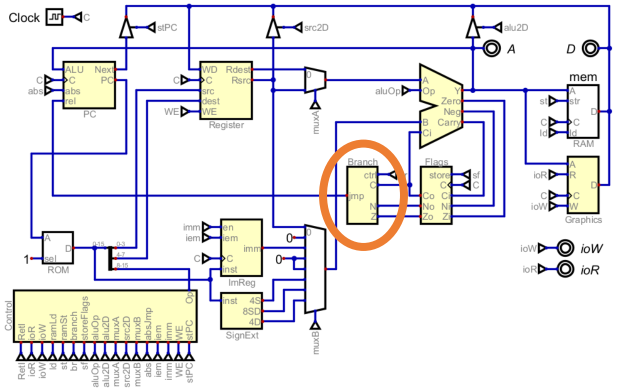


Other Components

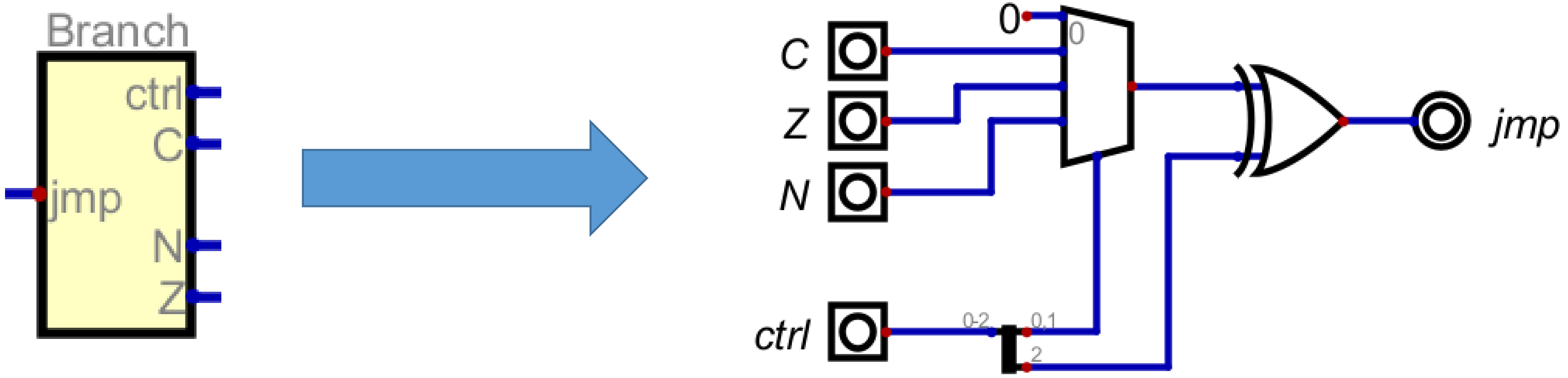
- This component handles jump and **branch** instructions.

- ctrl: Controls the type of branch
- C: Carry (1)
- N: Negative (-1)
- Z: Zero (0)
- jmp: If 1, indicates a conditional relative jump



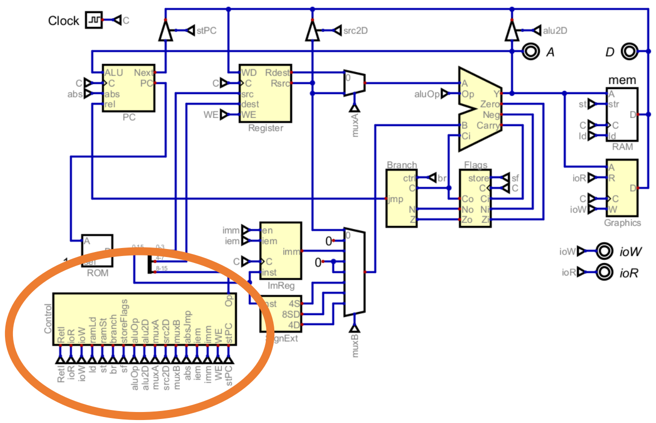


Other Components



Control Unit

- The last component, the control unit, is as fundamental to the processor as the ALU.
- The control unit activates different operations based on the supplied opcode



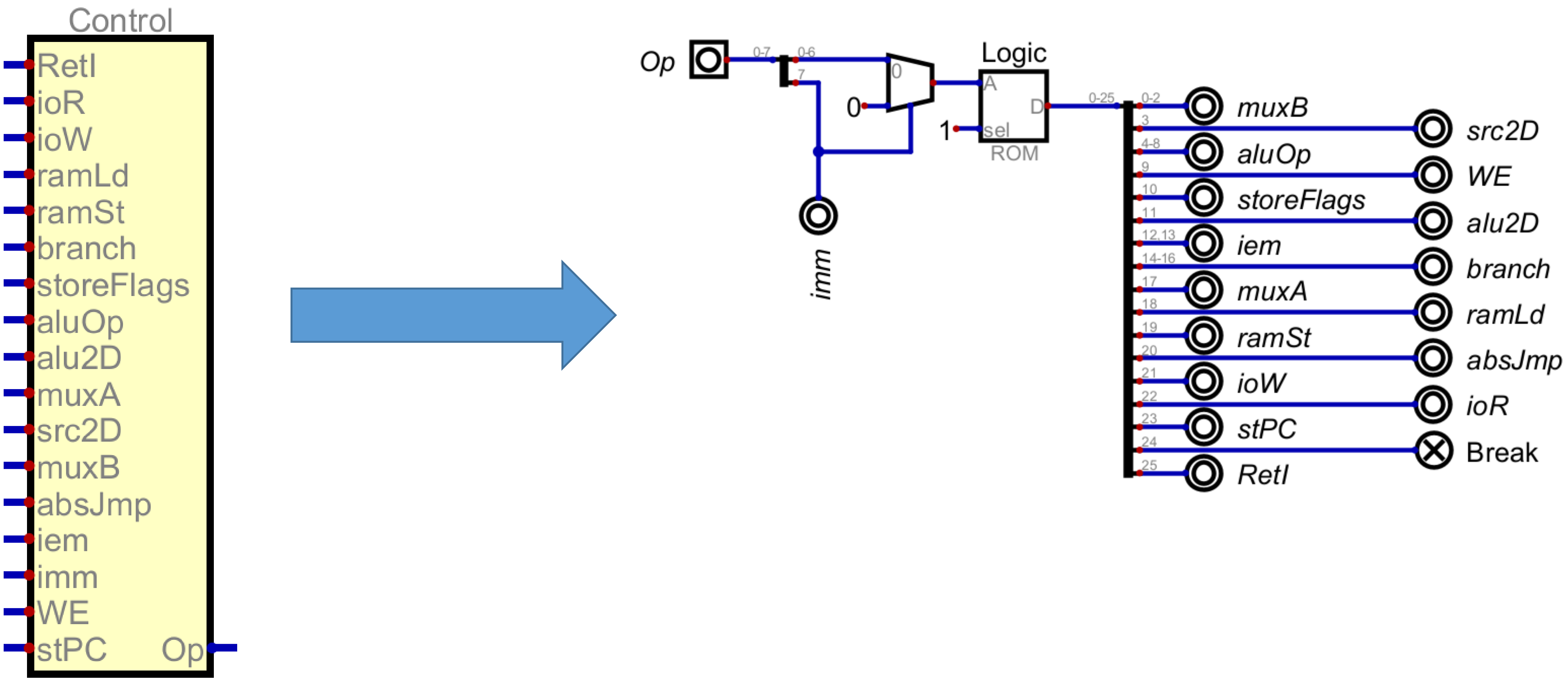
Control Unit

- RetI: Return from interrupt
- ioR: Read from I/O
- ioW: Write to I/O
- ramLd: Load data from RAM onto data bus
- ramSt: Store data to RAM from data bus
- branch: Type of branch
- storeFlags: ALU stores flags as a result of the operation
- aluOp: ALU's operation
- alu2D: ALU value is put on data bus

Control Unit

- muxA: Selector for multiplexer A
- src2D: Puts source register onto data bus
- muxB: Selector for multiplexer B
- absJump: Absolute jump
- iem: Immediate extend
- imm: Store a constant value
- WE: Value on the data bus is stored to a register
- stPC: Program counter is stored to a register

Control Unit



Processor Performance

- Almost all processors have a clock that determines when instructions are executed by it.

- These time intervals are measured in **clock cycles**
  - Also called clock ticks, clocks, or cycles

- Sometimes, the term **clock period** refers to
  - Complete clock cycle (usually measured in picoseconds; ps; 10^{-12} seconds)
  - Clock rate (the inverse of the clock cycle)

Processor Performance

- $Clock\ Cycle\ = \frac{second}{cycle}$

- $Clock\ Rate\ = \frac{cycle}{second}$

  - The unit for $\frac{cycle}{second}$ is Hertz (abbreviated Hz)

Processor Performance

- Example: What is the clock rate of a processor with a clock cycle of 250ps (picoseconds)?
  - $250ps = 250 * 10^{-12}s = 0.00000000025s$

- $Clock\ Cycle = \frac{250 \times 10^{-12} seconds}{cycle} = 0.00000000025 \frac{seconds}{cycle}$

- $Clock\ Rate = \frac{1\ cycle}{250 \times 10^{-12} seconds} = 4{,}000{,}000{,}000 \frac{cycles}{second} = 4.0 \text{ GHz}$

Processor Performance

- Example: What is the clock cycle of a processor with a clock rate of 3.8GHz?

- $Clock\ Rate\ =\ 3{,}800{,}000{,}000\ \frac{cycles}{second}$

- $Clock\ Cycle\ =\ \frac{1\ second}{3{,}800{,}000{,}000\ cycles}\ =\ 2.63\ \times\ 10^{-10}\,\frac{seconds}{cycle}\ =$

$263\ \times\ 10^{-12}\,\frac{seconds}{cycle}$ = $263\,\frac{picoseconds}{cycle}$ or simply $263\ ps$

Processor Performance

- Processor Performance can be measured in terms of **execution time**- How long it takes for a processor to run a program.

- CPU Execution time (seconds) = $clock\ cycles\ \times \boldsymbol{clock\ cycle\ time}$

- CPU Execution time (seconds) = $clock\ cycles\ \times \frac{\boldsymbol{seconds}}{\boldsymbol{cycle}}$

Processor Performance

- The clock rate could also be used, since it is the inverse of the clock cycle time.

- CPU Execution time (seconds) = $\dfrac{Clock\ cycles}{\boldsymbol{Clock\ rate}}$

- CPU Execution time (seconds) = $\dfrac{Clock\ cycles}{\dfrac{\boldsymbol{cycles}}{\boldsymbol{second}}}$

Processor Performance

- A program requires 10 x 10^9 cycles to complete. The program is executed by a 2GHz processor. How long will it take the processor to execute the program?

- CPU Execution time = $10 \;\times 10^9\; cycles \times \frac{1}{2 * 10^9} \frac{s}{cycles}$ = $\frac{10 \times 10^9}{2 \times 10^9}\, s$ = 5 s

- CPU Execution time = $\dfrac{10 \times 10^9\; cycles}{\dfrac{2 \times 10^9 cycles}{s}}$ = 5 s

Processor Performance

- It takes a 2GHz processor 15 seconds to execute a program. How many clock cycles did the program need in order to complete?
- 15 s = $\boldsymbol{x}\ cycles \times \frac{1}{2 \times 10^9} \frac{s}{cycles}$

$$\boldsymbol{x}\ cycles = \frac{15\ s}{\frac{1}{2\ \times 10^9} \frac{s}{cycles}} = 3\ \times\ 10^{10}\ cycles$$

Processor Performance

- It takes a 2GHz processor 15 seconds to execute a program. How many clock cycles did the program need in order to complete?
- $15\ s = \dfrac{\boldsymbol{x}\ cycles}{\dfrac{2 \times 10^9\ cycles}{s}} =$

$$\boldsymbol{x}\ cycles = 15\ \cancel{s}\ \times \frac{2 \times 10^9\ cycles}{\cancel{s}} = 3\ \times\ 10^{10}\ cycles$$

Processor Performance

- Processor Performance can also be measured in terms of *instruction performance*- or, the average number of clock cycles required for each instruction.
  - **CPI: C**ycles **p**er **i**nstruction

- Clock Cycles = $Number\ of\ instructions\ \times\ CPI$

Processor Performance

- A program has 50 instructions and is executed on a processor with a CPI of 1.5. How many clock cycles are required by the program?

- Clock Cycles = $50\ instructions\ \times\ 1.5\ \frac{cycles}{instruction}\ =\ 75\ clock\ cycles$

Processor Performance

- A program has 100 instructions and is executed on a 3GHz processor with a CPI of 1.5. How long will it take the program to execute?

- Clock Cycles = $100\, instructions \times 1.5\, \frac{cycles}{instruction} = 150\, clock\ cycles$

- Then use an execution time formula:

$$\frac{150\, cycles}{\frac{3 * 10^9\, cycles}{s}} = .00000005\, s = 5 \times 10^8\, s$$

Processor Performance

- The formula:

$$\boldsymbol{Clock\ Cycles\ = Instructions\ \times\ CPI}$$

- Can be substituted in the execution time formulas for a more general formula:

$$CPU\ Execution\ time\ =\ \boldsymbol{Clock\ cycles}\ \times\ clock\ cycle\ time$$

$$\boxed{CPU\ Execution\ time\ = \boldsymbol{Instructions}\ \times\ \boldsymbol{CPI}\ \times\ clock\ cycle\ time}$$

$$CPU\ Execution\ time\ =\ \frac{\boldsymbol{Clock\ cycles}}{Clock\ rate}$$

$$\boxed{CPU\ Execution\ time\ =\ \frac{\textbf{Instructions}\ \times\ \textbf{CPI}}{Clock\ rate}}$$

Processor Performance

- Using either, we can derive the number of instructions:

$$CPU\ Execution\ time\ =\ Instructions\ \times\ CPI\ \times\ clock\ cycle\ time$$

$$\boldsymbol{Instructions}\ =\ \frac{CPU\ Execution\ time}{CPI\ \times\ clock\ cycle\ time}$$

$$CPU\ Execution\ time\ =\ \frac{\text{Instructions}\ \times\ \text{CPI}}{Clock\ rate}$$

$$\boldsymbol{Instructions}\ =\ \frac{CPU\ Execution\ time\ \times\ clock\ cycle}{CPI}$$

Processor Performance

- Instructions Per Second (IPS)

$$IPS = \frac{Clock\ Rate}{CPI}$$

Amdahl's Law

- A program runs in 100 seconds and 80 seconds are spent performing multiplication operations.
- *How much does the speed of multiplication need to improve for the program to run 5 time as fast?*

- Amdahl's Law:

$$Time\ after\ improvement\ = \frac{Time\ affected\ by\ improvement}{Amount\ of\ improvement} + Time\ unaffected$$

Amdahl's Law

- A program runs in 100 seconds and 80 seconds are spent performing multiplication operations.
- *How much does the speed of multiplication need to improve for the program to run 5 times faster?*

$$Time\ after\ improvement\ =\ \frac{80\ s}{n} + (100 - 80)\ s$$

$$20\ s\ =\ \frac{80\ s}{n} + 20\ s$$

$$0\ s\ =\ \frac{80\ s}{n}$$

- No amount by which the speed can be increased 5 times.

Amdahl's Law

- A program runs in 100 seconds and 80 seconds are spent performing multiplication operations.
- *How much does the speed of multiplication need to improve for the program to run **2** times as fast?*

$$Time\ after\ improvement\ = \frac{80\ s}{n} + (100 - 80)\ s$$

$$50\ s\ = \frac{80\ s}{n} + 20\ s$$

$$30\ s\ = \frac{80\ s}{n}$$

$$n(30\ s) = 80\ s$$

$$n = 2.667$$

- Thus, multiplication operations must improve by 2.667 times