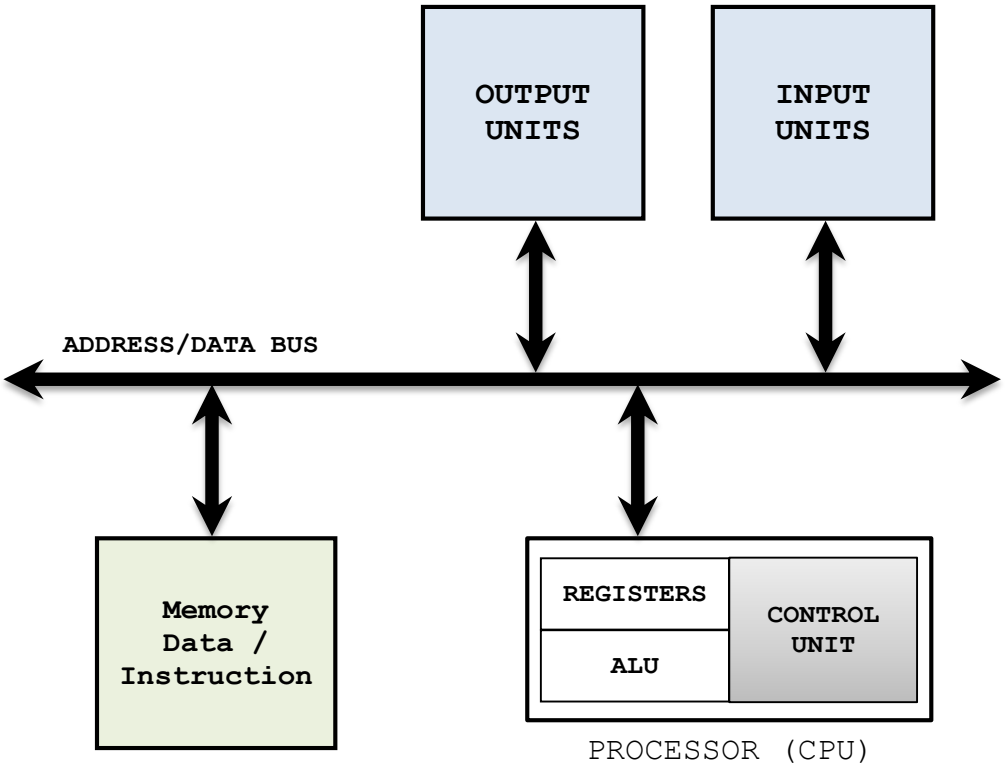
Notes – Unit 1

COMPUTER HARDWARE ORGANIZATION



THE PROCESSOR

Three major components:

- **Arithmetic Logic Unit (ALU)**: Performs arithmetic (e.g., addition, subtraction, division) and bit-wise logic (e.g., AND, OR) operations.
- **Registers**: They hold data and memory address values during the execution of an instruction.
- **Control Unit**: It is in charge of executing instructions. Instructions are read from memory. To execute a particular instruction, this unit asserts specific signals at certain times to control the registers, ALU, memories and ancillary logic. A Control Unit is implemented as a large Finite State Machine (FSM) with datapath logic.

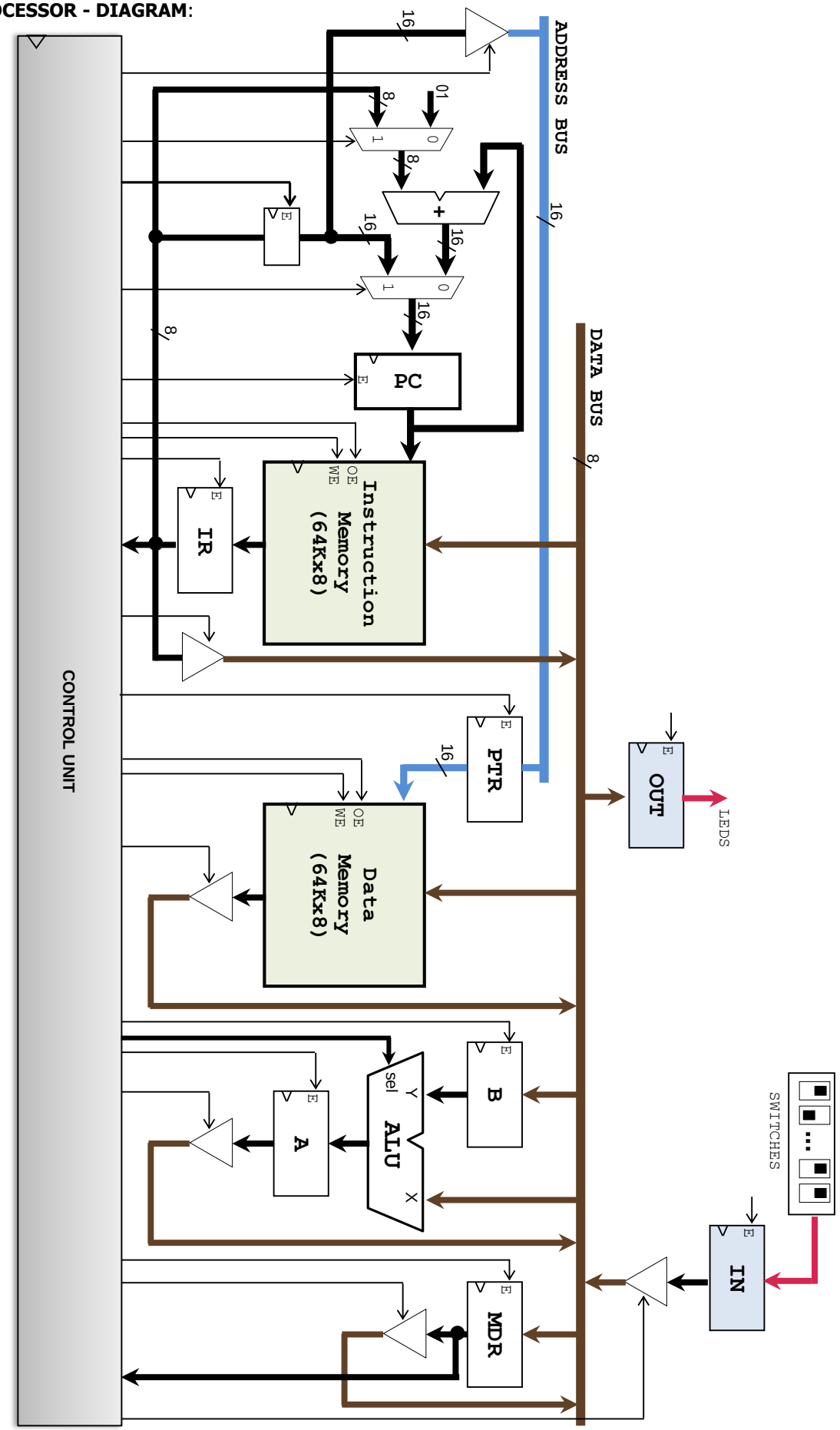
EXAMPLE OF A SIMPLE PROCESSOR

- Instruction (or Program) Memory and Data Memory separated (Harvard architecture).
- A handful of registers: A, PC, PTR, MDR, IR
- 8-bit processor: Data bus is 8-bits wide
- Data and Instruction memories: Memory contents are 8-bits wide. The size of the memories is 64 KB, this requires 16 bits to address all bytes. Address bus is then 16-bits wide.
- Program Counter (PC): Keeps track of address of the instruction to be executed next. It points to the Instruction Memory. It can be i) incremented by 1, ii) incremented by an 8-bit offset, iii) modified to an arbitrary 8-bit or 16-bit address.
- PTR Register (16 bits): Points to data memory.

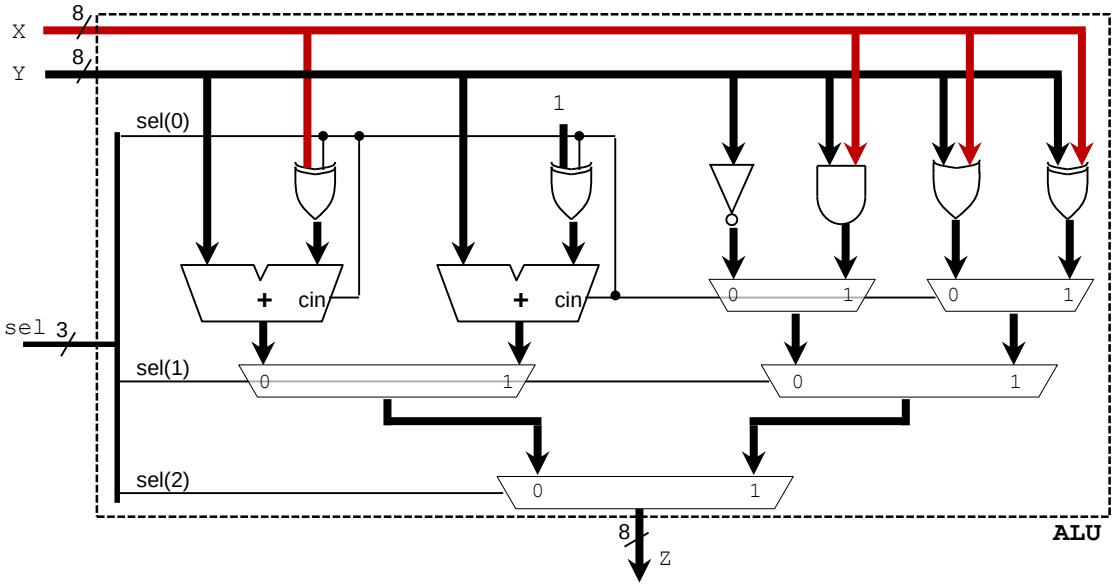
PROGRAM EXECUTION

- The processor reads instructions from the Instruction Memory (one after the other) and executes them. The instructions are a string of bytes that specify a particular operation to be carried out. The Control Unit asserts specific signals at the proper times so as to execute a particular instruction. The results of the operation can be stored in a register, data memory, and/or and output interface. Some instructions might modify the Program Counter at will.
- At power up, usually the processor reads the instruction from the first memory address. This is done by asserting the reset input of the Program Counter Register at power up.

SIMPLE PROCESSOR - DIAGRAM:



ARITHMETIC LOGIC UNIT:



sel	Operation	Function	Unit
000	Z <= X + Y	Add X and Y	Arithmetic
001	Z <= X - Y	Subtract Y from X	
010	Z <= X + 1	Increment X	
011	Z <= X - 1	Decrement X	
100	Z <= not(X)	Complement X	Logic
101	Z <= X and Y	Bitwise AND	
110	Z <= X or Y	Bitwise OR	
111	Z <= X xor Y	Bitwise XOR	

INSTRUCTION SET:

- The format of the instructions is as follows: OPCODE (1 byte) | (optional fields: up to 2 bytes)
  - ✓ OPCODE: Unique identifier that specifies a particular instruction.
  - ✓ Other fields: usually they are address and data values
- An instruction may take up to 3 bytes in memory.

Instruction Set: Collection of available instructions.

Assembly Instruction mnemonic	Machine code	Meaning
`ld addr,#val`	`75 aa xx`	Load 8-bit value (`val`) into memory location at `addr`.
`ld ptr, #data`	`90 yy yy`	Load 16-bit value (`data`) into register `ptr`.
`ld A, @ptr`	`E0`	Load contents of memory location pointed to by `ptr` into `A`.
`and A, #val`	`54 xx`	Bit-wise AND: `A ← A and val`
`bnz A, target`	`70 zz`	If `A≠0`, then branch to the address provided by target If `A=0`, then go to next instruction
`inc addr`	`05 aa`	Increments the contents of memory location at `addr`.
`dbnz addr, target`	`D5 aa zz`	Decrements the contents of memory location at `addr` and branch if the result is not zero.

Observations:

- Most of the instructions in the table above only allow 8 bits to specify a memory address.
- PTR is a 16-bit register. To load a 16-bit value on it, we require two read cycles from Instruction Memory. This is taken care of by an intermediate register that grabs 8 bits at a time.
- `bnz, dbnz`: A label specifies the instruction to branch to. The assembler determines the address to branch to and specifies it in the machine code (here we can only branch to the first 256 memory positions as only one byte is available to specify the address to branch to).
  Alternative assembler approach: the assembler determines an offset. This way we can branch (up or down) 256 positions from the current instruction. This approach requires specification of whether to add or subtract the offset from the PC value, thereby increasing the machine code and requiring the PC circuitry to allow for addition and subtraction.

SAMPLE PROGRAM:

Assembly Instruction mnemonic	Start Address	Machine Code (hex)	Comments
`ld 0x20,#0`	0x0000	75 20 00	
`ld 0x21,#20`	0x0003	75 21 14	20 is a decimal value
`ld ptr,#0x2000`	0x0006	90 20 00	
`loop: ld A, @ptr`	0x0009	E0	Only 1 byte required
`and A,#0x03`	0x000A	54 03	
`bnz A, next`	0x000C	70 10	
`inc 0x20`	0x000E	05 20	
`next: dbnz 0x21, loop`	0x0010	D5 21 09	

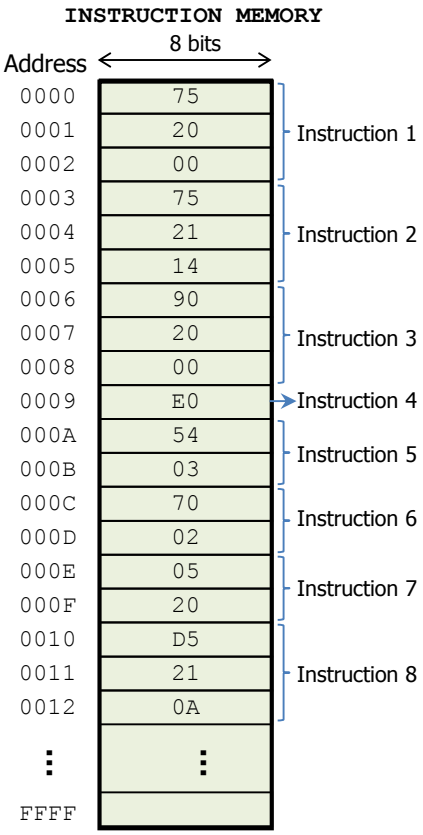
1) `ld 0x20,#0`: Load `0x00` at address `0x0020`. Instruction requires 3 bytes.
The OPCODE (`75`) is read from Instruction Memory. The next byte (`0x20`) is read and placed on PTR (left-appending zeros) via the address bus. The following byte (`0x00`) is read and placed on the data bus. Then we write `0x00` in Data Memory at address `0x0020` (alternatively, we can store `0x00` on MDR, this allows for flexibility to choose when to write on Data Memory).
Start of Instruction: PC: `0x0000` End of Instruction: PC: `0x0003`

2) `ld 0x21,#20`: load `0x14` at address `0x0021`. Instruction requires 3 bytes.
We write `0x14` in Data Memory at address `0x0020` (as in instruction 1)
Start of Instruction: PC: `0x0003` End of instruction: PC: `0x0006`

3) `ld ptr,#0x2000`: load `0x2000` on PTR. Instruction requires 3 bytes.
The OPCODE (`90`) is read from Instruction Memory. The next byte (`0x20`) is placed on the least significant byte of PTR. The following byte (`0x00`) is placed on the most significant byte of PTR.
Start of Instruction: PC: `0x0006` End of Instruction: PC: `0x0009`

4) `ld A, @ptr`: contents of the memory location pointed to by PTR are loaded into A. This instruction requires 1 byte.
The OPCODE (`E0`) is read from Instruction Memory. The data located at the address pointed to by PTR (`0x2000` in this example) is read and placed on the ALU where we transfer it to register A (this ALU transfer can be done by setting the register B to `0x00` and performing a bit-wise OR)
Start of Instruction: PC: `0x0009` End of Instruction: PC: `0x000A`

5) `and A,#0x03`: `A ← A AND 0x03`. This instruction requires 2 bytes.
The OPCODE (`54`) is read from Instruction Memory. The next byte (`0x03`) is placed on register B via the data bus. Then a bitwise AND with A is performed on the ALU, the result is stored in A.
Start of Instruction: PC: `0x000A` End of Instruction: PC: `0x000C`



6) `bnz A, next`: Branch to `next` (`0x0010`) if A is nonzero. This instruction requires 2 bytes.
The OPCODE (`70`) is read from Instruction Memory. The next byte (`0x10`) is read and placed on IR. To compare A with zero, A is placed on MDR. If A=0, PC is incremented by 1 (as usual). If A≠0, the value at IR (`0x10`) is placed on PC.
Start of Instruction: PC: `0x000C`
End of Instruction: If we do not branch (A is zero), then PC: `0x000E`. If we branch (A is nonzero), then PC: `0x0010`

7) `inc 0x20`: The data located in address `0x0020` is incremented by 1. This instruction requires 2 bytes.
The OPCODE (`05`) is read from Instruction Memory. The next byte (`0x20`) is placed on PTR, and we read the data at `0x0020`. The data is placed on the ALU, where it is incremented by 1 and placed in A. Then, the contents of A are written on the Data Memory at memory location specified by PTR (`0x0020`).
Start of Instruction: PC: `0x000E` End of Instruction: PC: `0x0010`

8) `dbnz 0x21, loop`: Decrement the data located in address `0x0021`. If the result is nonzero, branch to `loop` (`0x0009`)
The OPCODE (`D5`) is read from Instruction Memory. This instruction requires 3 bytes. The next byte (`0x21`) is placed on PTR. The next byte (`0x09`) is read and placed on IR. Then we read data at `0x0021` (pointed to by PTR). The data is placed on the ALU, (where it is decremented by 1 and placed in A. A is then placed on MDR to compare A with zero. If it is zero, we increment PC by 1 (as usual). If A is nonzero, the value on IR (`0x09`) is placed on PC.
Start of Instruction: PC: `0x0010`
End of Instruction: If we do not branch, then PC: `0x0013`. If we branch, then PC: `0x0009`