MS-101 Makerspace

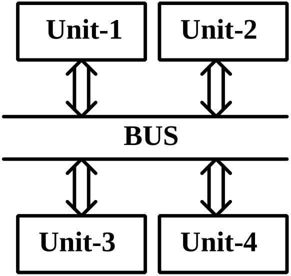
Introduction to Microprocessors

Department of Electrical Engineering
Indian Institute of Technology, Bombay

July 16, 2024

Connecting Digital Circuits Using a “Bus”

For Data communication between multiple units with digital outputs, it is convenient if we can connect their input/output ports using common wires.



This is called a **bus**.

However, outputs of conventional digital circuits cannot be shorted together.

Outputs of conventional digital circuits can only be ‘0’ or ‘1’. If we short the outputs of multiple circuits, some will try to make the output ‘High’ while the others may try to pull it ‘Low’.

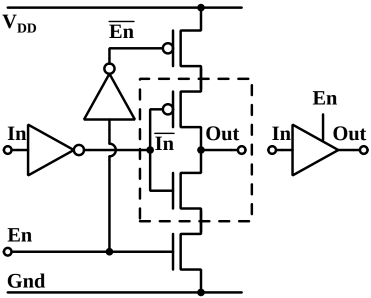
- This can lead to indeterminate outputs and heavy current being drawn from the supply.
- To connect outputs of multiple circuits to the same wire in the bus, we need circuits with **tri-stateable** outputs or **open-collector outputs**.

Tri-stateable Digital Circuits

- The output of tri-stateable digital circuits can have three states. It could be '0' or '1' like the conventional digital circuits, or it could be in a high impedance state called 'Z'.
- In the high impedance state, the circuit is **disconnected** from the output and does not interfere with other outputs connected to the same wire.

The figure on the right shows the circuit diagram and the symbol for a tri-stateable buffer.

The circuit has an extra input to enable it (– labelled as 'En' in the circuit). When En = 1, the circuit acts as a buffer, but when En = 0, the circuit is disconnected from the output.



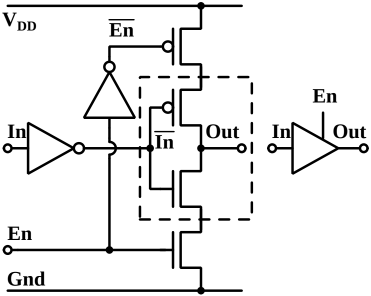
Tri-stateable Buffer Circuit

Circuits which can disconnect their output drivers from the output terminal are called **tri-stateable** circuits.

The connection to supply and ground is through the top pMOS and bottom nMOS transistors.

Both of these are OFF when En = 0 (and therefore $\overline{\text{En}} = 1$).

Both are ON when En = 1 and $\overline{\text{En}} = 0$, applying power to the circuit in the dashed box.



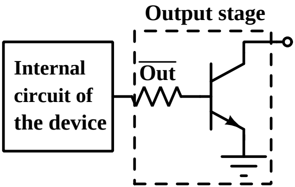
When En = 1, the middle two transistors in the dashed box form an inverter, which inverts the output of the first inverter, thus providing a buffer function.

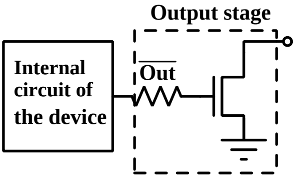
Bus Control with Tri-stateable Output Circuits

- One of the digital circuits on the bus acts as the "**bus master**".
- It decides which of the circuits on the bus will be **listeners** for the data on the bus and which will be the **talker**.
- While there can be multiple listeners on the bus, only one circuit can be a talker. The bus master manages this by setting the Enable line of the designated talker to TRUE and that of all others to FALSE.
- The bus itself contains many wires for carrying the data and a few additional wires driven by the bus master for bus administration functions such as designating roles as listeners or talkers.

Open Collector Outputs

Another way for direct connection of outputs to common wires is to use open-collector/open-drain output stages.

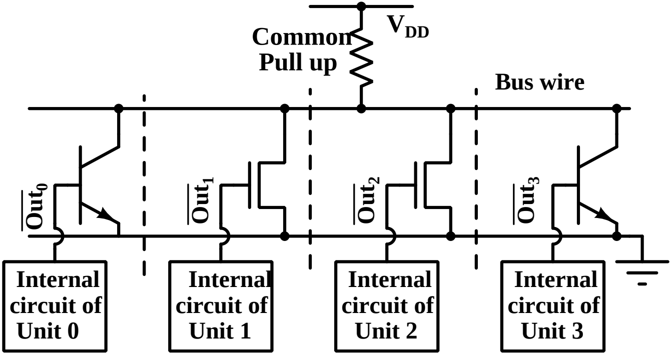




- The output stage of digital circuits normally includes a switching device for pulling the output up to the supply voltage, as well as a device to pull it down to ground.
- However, open collector/open drain circuits include only the pull down device.
- The collector/drain of the pull down device is brought out as the output terminal.

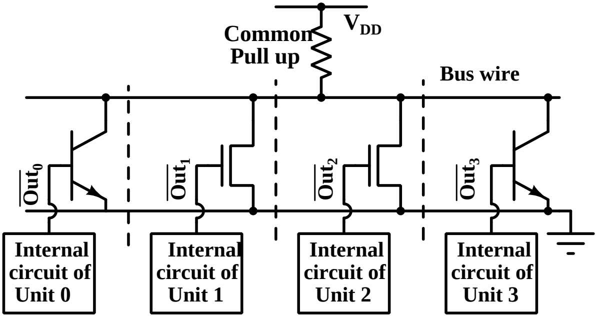
Outputs of open-collector/open-drain circuits can be shorted together. A common external resistor is used as a pull up for the shorted outputs.

Open Collector Outputs for Bus connection



- There is a common pull up resistor for all output devices which are effectively in parallel.
- Units turn OFF their output device to write a ‘1’ to the output and turn it ON to write a ‘0’ to the output.

Open Collector Outputs for Bus connection . . . contd.



Now it is possible for different units to output different logic values, and still have their outputs shorted.

- The bus wire will be pulled up by the common pull up resistor when **all** devices output a '1' (turning their output device OFF).
- The bus wire will be pulled down to '0' if **any** device outputs a '0' (turning its output device ON).
- A '0' is dominant over a '1' in this system. A device writing a '1' to the output is effectively disconnecting from the bus.

Need for a Programmable Digital Circuit

- Design of complex integrated circuits has a high fixed cost component.
- If sold in large quantities, the cost of an integrated circuit can be made quite low.
- It makes sense to design a programmable circuit which can be configured to perform different functions for different products.
- Now this circuit can be used for a variety of products and will thus be required in large quantities.
- The specific function performed by the circuit needs to be selected by a digital input. This is called an **instruction**.
- The circuit is given a series of instructions to perform a task. This sequence of instructions is a **program**.

The Processor

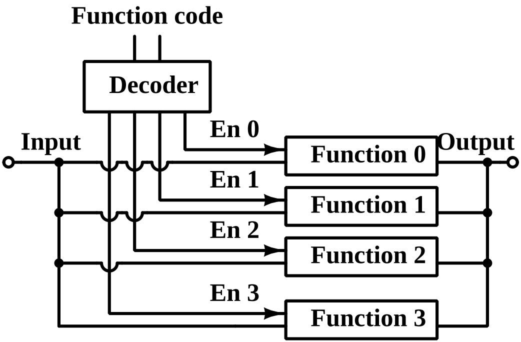
- The programmable digital circuit which can run a program is called a processor.
- A processor which is implemented on a single chip using VLSI technology is called a **microprocessor**.
- Modern microprocessors can be quite complex – a Pentium processor used in PCs has upwards of 100 million transistors!
- The basic logic in a processor just performs the following loop endlessly:
  - Fetch next instruction
  - Decode it
  - Execute the instruction

Notional Plan for a Programmable Digital Circuit

A notional plan for making a programmable circuit is shown below.

The common input going to the inputs of all functional blocks is fine, but the common output for these will cause problems if function outputs use conventional digital circuits!

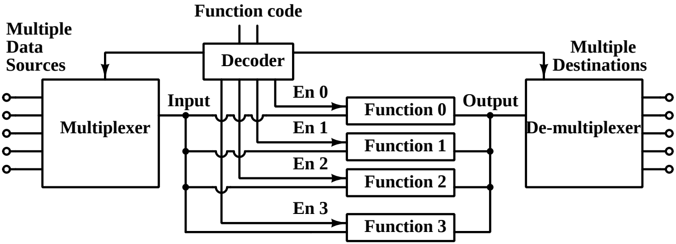
We should use tri-stateable outputs to share a common output terminal.



However, each tri-stateable circuit needs an individual enable signal. These enable signals can be provided by the decoder if these are included in the function code.

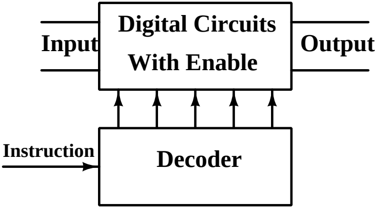
A Multipurpose Digital Circuit

Assume that we have multiple digital circuits, each performing a different function. A circuit can be selected using its enable input.



- We need a **multiplexer** to route the data from multiple sources to the input of this circuit and a **de-multiplexer** to route its outputs to a selected device.
- The multiplexer and de-multiplexer will also need ‘select’ inputs.
- All this information has to be supplied through the function code to configure this circuit to perform a selected function.

Choosing a Specific Sub-circuit

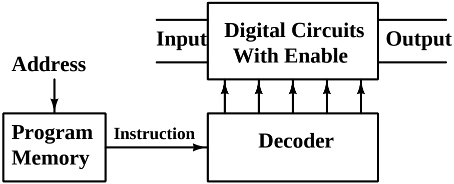


A lot of control inputs will be required to choose and configure a specific circuit and to route the data from input through this circuit to the output.

- We can encode this information in a compact form and use a decoder to generate all the detailed control signals necessary for data routing and for enabling circuits.
- This encoded information is called an **instruction.**
- The format of this information is decided by the circuit designer and the decoder expands out the control signal according to this format.

Storing Instructions

The actual task performed by this flexible circuit may require a *series* of operations to be carried out.

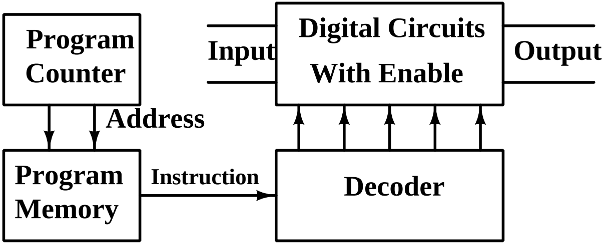


- For example, we may wish to multiply a variable by a coefficient and then add the product to an accumulating sum.
- This requires a group of instructions to be executed in a sequence.

This sequence of instructions is called a **program**.

We store the sequence of instructions in a memory called the *program memory*.

Fetching Instructions

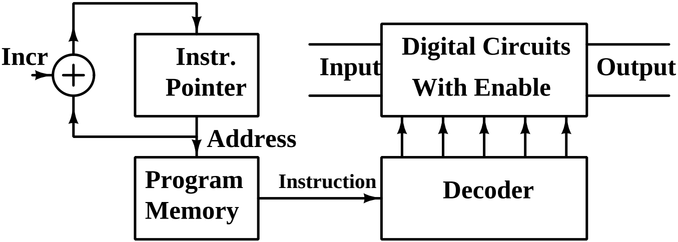


To fetch each instruction, we need to provide its address to the memory.

From where will this address come?

- We can include a counter to generate sequential addresses for the program memory.
- Every time an instruction is fetched from the memory, the counter will increment the address to point to the next instruction.
- The counter which provides addresses to the program memory is called a **program counter**.

Instructions From Auto-incremented Addresses

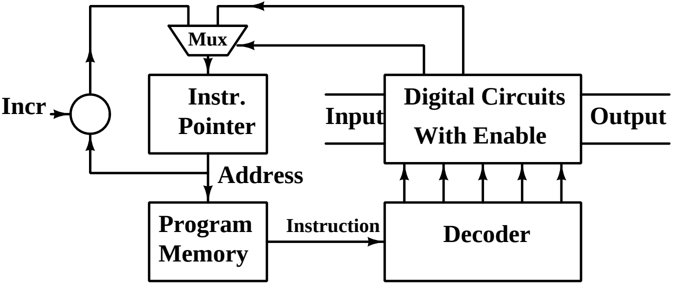


Fetch - Decode - Execute

- The program counter is just a register to store a value and an adder to increment the value stored in the register.
- This register is also referred to as the **instruction pointer**.

Instructions from Non-sequential Addresses: "Jump"

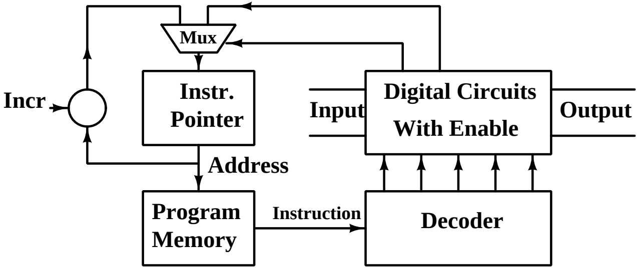
The instruction pointer needs to be loaded with some value when the whole operation starts.



- Instructions need not be executed in a strict sequence. After finishing a block of instructions, we may wish to execute a different block of instructions, stored at some other address.
- The ability to load data into the instruction pointer gives us the ability to alter the flow of instructions in the program during its run time.
- When a new value is loaded in the instruction pointer, the next instruction will be fetched from that new location.

Executing a “Jump” Instruction

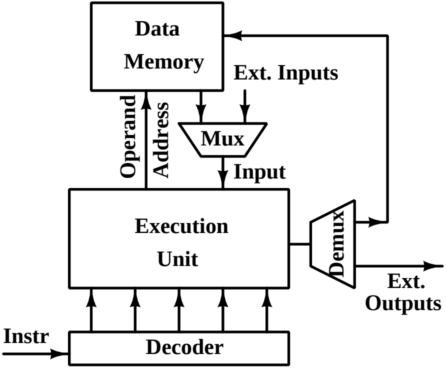
After finishing a block of instructions, we may wish to “jump” to a different block of instructions, stored at some other address.



- Moving to a different block of instructions may be accomplished by loading a different value into the instruction pointer.
- The value to be loaded into the instruction pointer can be provided by the last instruction of the first block.
- execution of this instruction will cause the multiplexer to choose this new address, rather than the default incremented address as the input to the instruction pointer.

Managing Data

So far we have concentrated on the flow of instructions and control path. What about the data path?



Different instructions will, in general, operate on different data.

How do we propose to manage the flow of data?

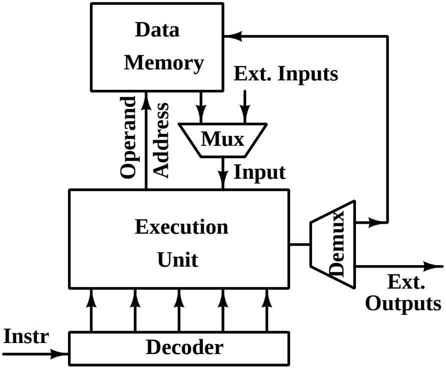
→ Data can also be stored in a memory. Successive data items can then be fetched from/written to this memory.

Managing Data . . . contd.

- For fetching and storing data, We shall need the address where the data item is to be fetched from/written to.
- The actual address of the data item is often stored in a register. (This register is called a data pointer).
- The instruction should identify the register which stores the address.

The circuit we are designing should also be capable of fetching data from *external sources* (other than the data memory) and also to output data to destinations other than the data memory.

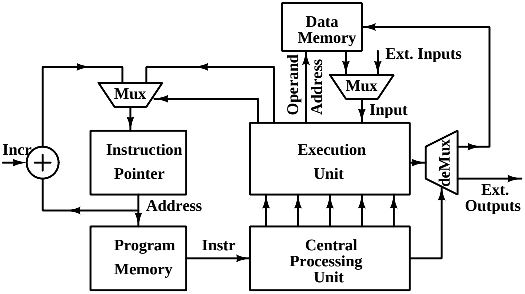
Data I/O with External World



- We can place a multiplexer in the input data path and a de-multiplexer in the output data path in order to choose between data memory and external IO.
- Control for these multiplexers/de-multiplexers should also be included in the instruction.
- The data and program memory need not be physically distinct.
- Different architectures choose either a common memory and data path for both instructions and data, or separate data and instruction memories with independent paths.

The Central Processing Unit

We have oversimplified the evolution of this circuit by assuming that just a combinational circuit like a decoder is needed to execute instructions.



Fetch - Decode - Execute

- In fact each instruction may need a sequence of actions to be executed properly.
- Therefore, we need a sequential circuit like a finite state machine to interpret and carry out an instruction.

- The circuit which generates the sequence of control inputs to the execution unit is called the Central Processing Unit or CPU.

The Central Processing Unit . . . contd.

- The central processing unit generates control inputs for the sequence of operations to be carried out.
- At the top level, it repetitively carries out the three steps:
  - Fetch the next instruction
  - Decode the instruction
  - Execute it
- Each of these operations may involve a series of transactions with the memory or with internal execution units.
- Instruction execution may involve data processing or controlling the flow of instruction through writing to the instruction pointer.

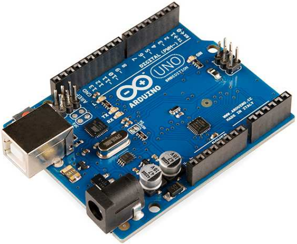
We Have Designed a Microprocessor! How to Use It?

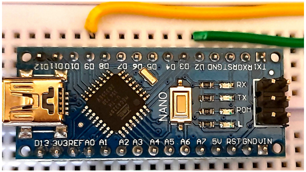
- The architecture which has evolved through this discussion describes the skeleton of a microprocessor.
- However, we need many devices external to the microprocessor to make a useful system – For example, program and data memories, display device drivers, keyboard readers etc.
- So with microprocessors, we invariably need multiple ICs on a circuit board to perform any useful function.
  For example, the old 8085 processor needed a program memory chip (say, type 2764, 8Kx8 EEPROM), data memory chip (type 6264, 8Kx8 RAM), 3 to 8 decoder (74LS138) to select I-O devices, data transceiver etc. in order to make an operational circuit.

Microcontroller Based Systems

- As technology has progressed, it has become possible to put many of the components required in the system along with the microprocessor on the same IC.
- Processors which have these components on the same IC are called **microcontrollers**.
- A system using microcontrollers can be quite compact and can be made operational in practical systems with very few external components.
- Ready-made cards with microcontrollers and the essential external components are available with varying capabilities.
- “Raspberry Pi” is an example for such cards and uses the popular ARM processor. This is a 32 bit or 64 bit system and is used in somewhat high end applications.

Microcontroller Based Systems . . . contd.





- “Arduino” is a family of cards whose design is open source. These are therefore made by several manufacturers and are quite economical.
- There are many models of Arduino cards. The most popular among these are UNO R3 and nano, which are based on the 8 bit AVR microcontroller ATMega 328P.
- These cards are suitable for low-cost and simple applications. The UNO R3 card can be bought for under Rs. 500, while the nano card is even cheaper.

Microcontroller ATMega 328P

- Arduino R3 and nano boards use the microcontroller ATMega 328P.
- This microcontroller contains:
  - an 8 bit microprocessor with 32 registers,
  - 32KB of flash memory to store the program,
  - 1 KB Electrically Erasable and Programmable memory,
  - 2 KB SRAM to store data,
  - 3 Counter/Timers,
  - 10 bit A to D converter with 6 input channels,
  - Serial communication using RS232C, I^2C as as well as SPI protocols.

The Arduino **board** adds a USB controller to the microcontroller resources to provide USB connectivity.

Application Development with Microcontroller Boards

Having selected a microcontroller board with the necessary resources, we need to program it to perform the function we need. As discussed earlier, this can be done using instructions recognized by the microcontroller.

- Instruction to the microcontroller are groups of '1's and '0's – which do not make much sense to human beings. These instructions are said to be in **machine code**.
- Meaningful mnemonics are associated with each machine code instruction which indicate the purpose for that instruction.
  For example, we associate MOV R31, R0 with a pattern like:
  0010 1101 1111 0000 for an instruction which copies the contents of register number 0 to Register no. 31.
- The collection of such mnemonics are then "assembled" into machine code by a program (typically run on an external computer).
- This program is called an **assembler**.

Application Development with Microcontroller Boards

. . . contd.

- While assembly language programs make sense to human beings, these are very detailed and it is laborious to write programs in assembly language.
- High level languages (such as C and C++) can express what needs to be done much more compactly.
- It is then possible to translate this high level language program to assembly language and eventually to machine language using a program called a **compiler**.
- Once the translation is complete, the bit pattern corresponding to the machine language must be downloaded to the microcontroller. This requires a communication program to run on the external computer and a receiver program to run on the microcontroller.
- The receiver program is a small program which is pre-loaded into the program memory of the microcontroller at the time of manufacture. This is called **boot code**.

Application Development with Microcontroller Boards

. . . contd.

The application development cycle involves

1. Writing the application program in a high level language (For example, C or C++).
2. Compiling it on the PC – usually using an Integrated Development Environment (IDE) supporting the board we shall be using. (The Integrated Development Environment includes an editor for writing the program, a compiler, a library with useful pre-written functions and the software required to send the machine code to the microcontroller board).
3. Downloading the compiled program to the board using IDE on the PC and the boot loader on the microcomputer.
4. Running the downloaded program on the board after resetting the processor.
5. Testing and debugging the application program.

Still to Come . . .

In the next lecture, we'll discuss the application development process with a few examples.