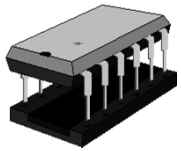


Simple CPU Operation and Buses

Dr Eddie Edwards
eddie.edwards@imperial.ac.uk

https://www.doc.ic.ac.uk/~eedwards/compsys

Heavily based on materials by Dr. Naranker Dulay

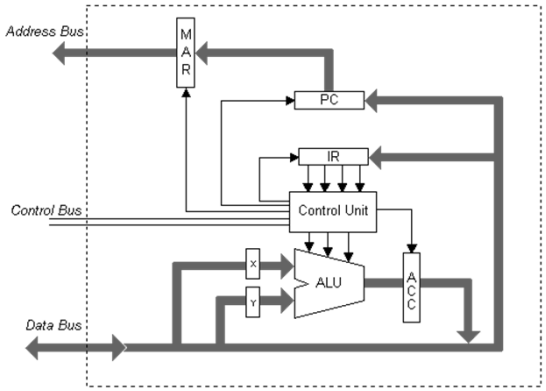
Learning Outcomes

- At the end of this lecture you will
  - Understand how a CPU might be put together
  - Be able to name the basic components - ALU, Registers, Control Unit
  - Be able to explain their function
  - Know the function of some special registers – program counter, instruction register and accumulator
  - Be able to describe the fetch-execute cycle
  - Know what a bus is and the difference between synchronous and asynchronous
  - Understand the Von Neumann architecture
  - Know how parallel architectures can be put together – e.g. GPU

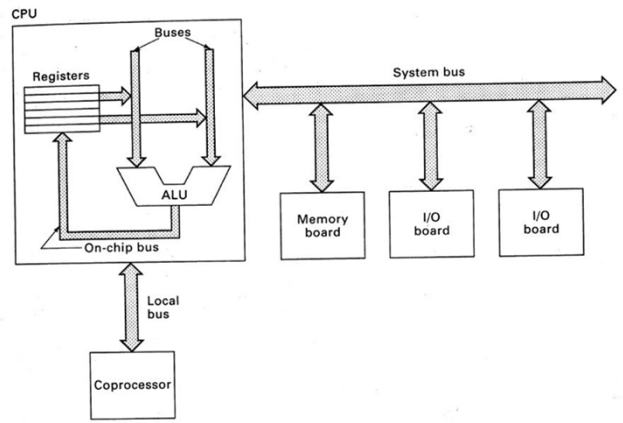
Composition of a CPU

- Control Unit
  - Generates control/timing signals
  - Decides which way data is going (in/out)
  - Controls decoding/execution of instructions
- Arithmetic Logic Unit
  - Execution of instructions
  - Mathematical operations (* / + - etc.)
  - Logical Operations: and/or/xnor etc as well as shift/rotate
- Registers
  - Small amount of very fast memory
  - Program counter – address of next instruction to be executed
  - Instruction register – Holds instruction while it is decoded
  - Accumulator – holds result of ALU operations
  - Other general purpose registers (e.g. stack pointer)

A simple CPU design



Computer will have several buses



Computer Buses

- Common electrical pathway between multiple devices.
- Internal to the CPU to transport data to and from the ALU
- External to the CPU to connect it to main memory and I/O controllers
- The **Bus Protocol** is a well-defined set of rules governing how the bus works
  - Including mechanical and electrical specifications
  - All attached devices must obey these rules
  - Makes it possible for boards designed by third parties to be attached

Example buses

- Multibus (Intel - 8086)
- IBM PC (PC/XT)
- ISA bus (PC/AT)
- EISA bus (80386)
- Microchannel (PS/2)
- PCI bus (Many PCs)
- Nubus (Macintosh)
- Universal Serial Bus (modern PCs)
- FireWire (Apple standard for high throughput – video/data)

Buses – Masters and Slaves

- Active devices attached to the bus that can initiate bus transfers are called **masters**
- Passive devices that wait for requests are called **slaves**
- Some devices may act as slaves at some times and masters at others
- Memory can never be a master device.

Examples of Bus Masters and Slaves

Master	Slave	Example
CPU	Memory	Fetching Instructions/Data
CPU	I/O	Initiating data transfer
CPU	Coprocessor	Handing off floating point operation
I/O	Memory	Direct Memory Access (DMA)
Coprocessor	Memory	Fetching operands

Properties of buses

- The binary signals that computer devices output are frequently not strong enough to power a chip.
- The bus may be relatively long or have several devices attached to it.
- Most bus masters are connected to the bus by a chip called a bus driver which is essentially a digital amplifier.
- Most slaves are connected to the bus by a bus receiver.
- For devices which can be both master and slave, a device called a bus transceiver is used.

Properties of buses

- These bus interface devices are often tri-state devices to allow them to disconnect ("float") when they are not needed.
- A bus has address, data, and control lines, but there is not necessarily a one-to-one mapping between CPU pins and bus lines. A decoder chip between CPU and bus would be needed in this case.

Bus address lines

- The more address lines a bus has, the more memory the CPU can address directly. If a bus has n address lines, then the CPU can use it to address 2^n different memory locations.
- Wider buses are more expensive:
  - Need more wires
  - Need larger connectors
  - Take up more space on the motherboard
- Early PC buses soon did not contain enough address lines, leading to various backward-compatible upgrades to the bus.

Bus data lines

- The number of **data lines** needed also tends to increase over time.
- Two ways to increase the throughput of a bus:
  - decrease the bus cycle time
  - increase the data bus width
- Speeding up the bus can result in problems of "bus skew" since data on individual lines travel at slightly different speeds.
- Better to increase the data width
- e.g. the (IBM-)PC went
  - from 8 data lines (PC bus)
  - to 16 data lines (AT/ISA bus)
  - to 32 data lines (EISA bus)
  - Each backward-compatible with its predecessor. When the PCI bus was first introduced on the PC, the motherboards also kept two or three EISA slots to enable use of "legacy" cards.
- Could use a **multiplexed bus**
  - Breaks up the bus operation into multiple steps
  - Same lines used for both data and addressing
  - Slows down bus performance

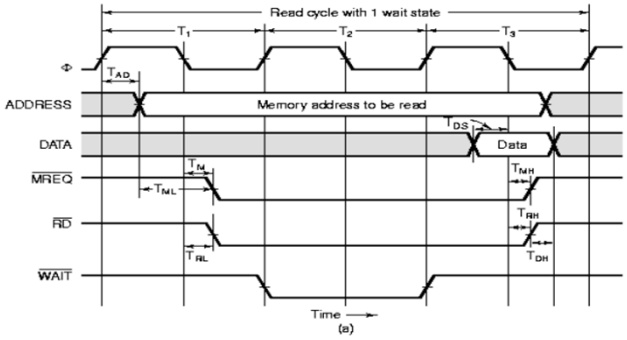
Bus synchronisation

- Buses can be divided up into two categories, depending on their synchronisation:
- **Synchronous bus**
  - has a line driven by a crystal oscillator (Master Clock)
  - The signal on this line consists of a square wave with a frequency of 5 - 100 MHz
  - All bus activities take an integral number of these cycles, called bus cycles
- **Asynchronous bus**
  - No master clock
  - Bus cycles can be of any length required and need not all be the same

Synchronous bus

- **Example**
  - A synchronous bus with a 40-MHz clock, gives a clock cycle of 25 nsec.
  - Assume reading from memory takes 40 nsec from the time the address is stable.
  - It takes three bus cycles to read a word.
  - MREQ' indicates that memory is being accessed
  - RD' is asserted for reads and negated for writes
  - WAIT' inserts wait states (extra bus cycles) until the memory is finished

Synchronous bus – memory read timings



Note – greyed out values are unimportant

Critical timings

Symbol	Parameter	Min	Max	Unit
T_{AD}	Address output delay		11	nsec
T_{ML}	Address stable prior to $\overline{MREQ}$	6		nsec
T_{M}	$\overline{MREQ}$ delay from falling edge of Φ in T_1		8	nsec
T_{RL}	$\overline{RD}$ delay from falling edge of Φ in T_1		8	nsec
T_{DS}	Data setup time prior to falling edge of Φ	5		nsec
T_{MH}	$\overline{MREQ}$ delay from falling edge of Φ in T_3		8	nsec
T_{RH}	$\overline{RD}$ delay from falling edge of Φ in T_3		8	nsec
T_{DH}	Data hold time from negation of $\overline{RD}$	0		nsec

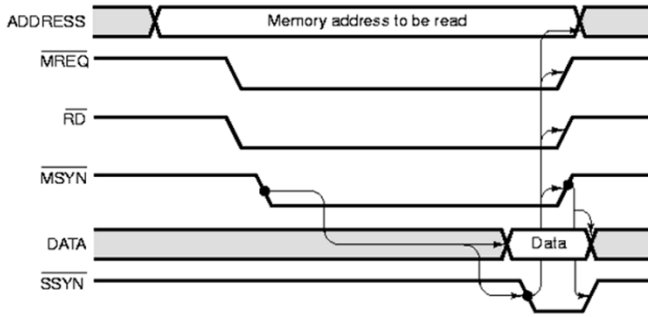
Synchronous bus

- Although synchronous buses are easy to work with due to their discrete time intervals, they also have some problems.
- Everything works in multiples of the bus cycle.
- If the CPU and memory can complete a transfer in 3.1 cycles they have to stretch it to 4.0 because fractional cycles are impossible.
- Once a bus cycle has been chosen, and memory and I/O cards have been built for it, it is difficult to take advantage of future improvements in technology. The bus has to be geared to the slowest ("legacy") devices on the bus.

Asynchronous bus

- Enables handling of "mixed" technology.
- The master device asserts MREQ', RD', etc. and then asserts MSYN (Master SYNchronization)
- Seeing this, the slave device starts its work
- When it is finished it asserts SSYN' (Slave SYNchronization)
- Seeing this, the master reads the data from the slave
- When it is done, it negates MREQ', RD', the address lines, MSYN' and SSYN'
- This completes the read, with all signals back to their original state

Asynchronous bus operation



Asynchronous bus operation

- The essential operation consists of four events:
  1. MSYN is asserted
  2. SSYN is asserted in response to MSYN
  3. MSYN is negated in response to SSYN
  4. SSYN is negated in response to the negation of MSYN
- A set of signals that interlocks in this way is called a **full handshake**. This is independent of timing. Each event is caused by a prior event, not by a clock pulse. If a particular master-slave pair is slow, that in no way affects a subsequent master-slave pair that is much faster.

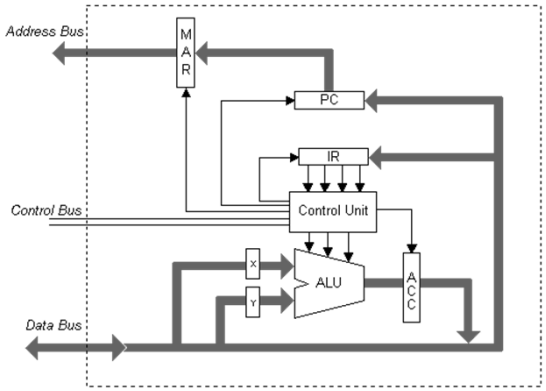
Bus synchronisation

Despite the advantages of asynchronous buses, **most buses are synchronous** since it is easier to build a synchronous system. The CPU just asserts its signals, and the memory just reacts. There is no feedback (cause and effect), but if the components have been chosen properly, everything will work **without handshaking**.

Bus operation – further issues

- If more than one device wants to become bus master, a decision must be made – know as **bus arbitration**
- May be **centralised** – an arbiter makes the decision (often the arbiter is part of the CPU)
- In **decentralised arbitration** there is no arbiter. Priority is decided by the design of the bus
- Often the CPU will have low priority, as timing is more important on external devices (e.g. a spinning disk)
- Another important kind of bus cycle is for handling **interrupts**.
  - When the CPU commands an I/O device to do something, it usually expects an interrupt when the work is done.
  - Interrupts may also be generated by the I/O device (mouse/keyboard/disk etc)
  - The interrupt signalling requires the bus.

Back to the CPU design



Composition of a CPU

- Control Unit
  - Generates control/timing signals
  - Decides which way data is going (in/out)
  - Controls decoding/execution of instructions
- Arithmetic Logic Unit
  - Execution of instructions
  - Mathematical operations (* / + - etc.)
  - Logical Operations: and/or/xnor etc as well as shift/rotate
- Registers
  - Small amount of very fast memory
  - Program counter – address of next instruction to be executed
  - Instruction register – Holds instruction while it is decoded
  - Accumulator – holds result of ALU operations
  - Other general purpose registers (e.g. stack pointer)

Fetch-Execute Cycle

- Fetch the ***Instruction***
- Increment the ***Program Counter***
- Decode the ***Instruction***
- Fetch the ***Operands***
- Perform the ***Operation***
- Store the ***Results***
- **Repeat** Forever

High-Level/Low-Level Languages, Machine Code

- High-Level Language (e.g. Java, C++, Haskell)

```
A = B + C                    Assignment Statement
```

- Low-Level Language -> Assembly Language (e.g. Pentium, PowerPC, ARM etc, Java Bytecode)

```
LOAD   R2, B          Assembly Language
ADD    R2, C          Instructions
STORE  R2, A
```

- (Binary) Machine Code

```
0001101000000001      Machine Code
0011101000000010      Instructions
0010101000000000
```

Very simple animated CPU

Fetch-execute cycle - timings

- Fetch Time depends on
  - Access time of main memory
  - Activity on System Bus
- Decode/Execute Time depends on
  - Speed of System Clock (Cycles per second - MHz)
  - Type of instruction

Von Neumann Architecture

- CPU can only execute One Instruction at a Time
- Instruction operates on only one set of operands at a time
- CPU Executes a single sequence of instructions which operate on a single stream of data
- Also known as "Single Instruction, Single Data stream" (SISD) machine
- Instructions and Data can both be stored in Main Memory (no distinction)
- Output of algorithms is deterministic (same input, same output)

Microprocessors

- 8-Bit µP (1970s)
- Zilog Z80 • Motorola 6800 • NS 6502
- 16-bit µP (early 1980s)
- Intel 8086/88, 80286 • Motorola 68000
- 32-bit µP (mid 1980s)
- Motorola 68030/40 • Intel 80386
- ...1990s: 64-bit machines e.g. DEC Alpha and now most PCs
- Internal Architecture may be different from size of Data Bus
- Intel 8088: internally 16-bit, 8-bit data bus
- Intel 386sx: internally 32-bit, 16-bit data bus

Alternative architectures

- Parallel
  - Most PCs have multiple CPUs (e.g. quad core)
  - Applications will split tasks in to multiple threads
  - It is not known in advance which thread will finish first
  - Can use signalling to resolve (similar to bus handshaking) – e.g. semaphores
- Graphics Processing Unit (GPU) Architecture
  - Massively parallel
  - Individual processors (pixel/vertex shaders) designed for very specific tasks
  - Ideal for processing of graphics on a pixel-by-pixel or vertex-by-vertex basis
  - General purpose GPU (GPGPU) can be used for other purposes
  - Generally GPUs do not care about which process finishes first
  - Result of GPGPU algorithms can be non-deterministic
- CPU and GPU are interdependent and converging

Summary

- Composition of a CPU
- Buses: Internal, Address, Data, Control
  Bus synchronisation (synchronous/asynchronous)
  Arbitration and interrupt handling
- Fetch-Execute Cycle
- CPU Components: Registers, ALU, Control Unit
- Registers: General Purpose Registers, Program Counter (PC), Instruction Register (IR), ALU Registers

Understand how data can be moved around and processed by a CPU