ENS1161 Computer Fundamentals

Module 2
Basic Computer Architecture and Principles of Operation

Unit Learning Outcomes

On completion of this unit, students should be able to:

- Describe the fundamental architecture and operating principles of a computer system.
- Interpret computer system specifications and standards and how they relate to system function.
- Compare different types of components and subsystems and their relative impacts on system function and performance.
- Make recommendations on the suitability of computer systems and components for a given function.
- Explain the interconnection between the software and hardware components of computer systems, and the processes involved in making them work together.

Unit Topics

Module	Topic
1	Introduction and overview
2	Basic Computer Architecture and Principles of Operation
3	Data and instruction formats
4	Basic computation in computers
5	Programming languages and tools
6	Memory devices and storage systems

Module	Topic
7	I/O devices and interfacing
8	I/O modes and BIOS
9	Operating Systems
10	Networks and the Internet
11	Embedded Systems and Cloud Computing
12	Review & Revision

Module Objectives

On completion of this module, students should be able to:

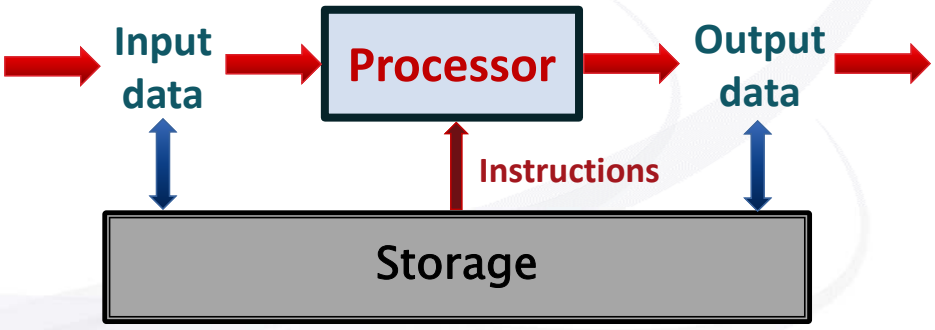
- Name the major components of a computer system and explain their function.
- Describe the processes and components involved in the fetch, decode and execute portions of an instruction cycle.
- Explain the difference between and operation and operand and how they are used by a processor.
- Explain what an instruction set is and how a processor is able to execute various instructions.
- Describe processor development trends and the factors affecting processor performance.

Introduction

- **Topic Scope**
  - The components of a computer
  - Basic internal architecture of a processor
  - Principles of operation of a processor
  - Illustration of a simple program execution using a hypothetical processor

Basic Computer System *(recap Module 1)*

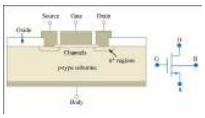
- A computer is essentially a system that takes input data and processes it to produce output data

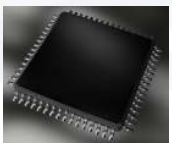


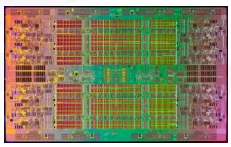
- Storage is used to hold the data and instructions for the processor

What is a processor? *(recap Module 1)*

- Fundamentally: a programmable logic device
- What can it do?
  - At simplest level - can turn signals on or off according to pre-specified conditions (like a switch)
  - Made up of lots of tiny electronic switches (transistors) connected together
- Used as a data processor
- Or applied to process control or system automation tasks

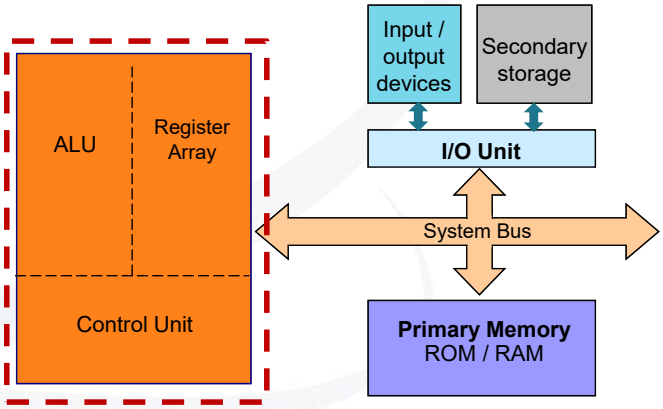






Basic Components of a Computer

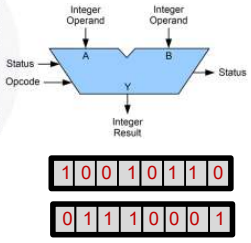
- Every computer contains the same basic components:
  - Arithmetic logic unit (ALU)
  - Register array
  - Control unit
  - Memory
  - Input/Output (I/O) unit
  - System Bus

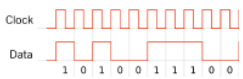


'Processor'

Basic Components of a Computer

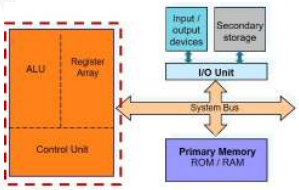
- The first 3 comprise the actual basic processor:
  - **ALU** – Performs arithmetic and logic operations on data stored within the processor in the …
  - **Register array** – Registers used for temporary data storage within the processor
  - **Control unit** – Provides the control and timing signals to all other components within the microprocessor system
    - Controls the flow of data through the system
    - Interprets and executes instructions
    - Synchronise the actions of various components through clock signals
  - Current processors also integrate many more functions





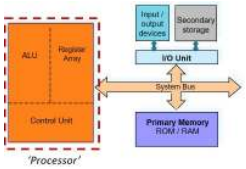
Basic Components of a Computer

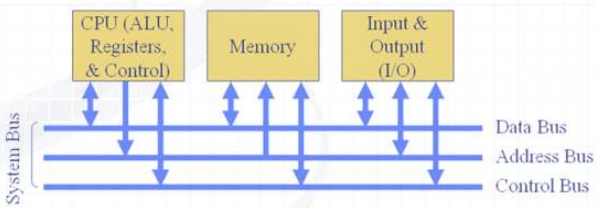
- On its own, a processor is of little use
- Need 3 other components as part of the system:
  - **Primary Memory**
    - Stores program instructions and data
    - provides that information to the processor whenever required
    - read only (ROM) or read/write (RAM)
  - **Input/Output unit**
    - Allows data/signal transfer between 'external' devices and the processor/memory
      - Including secondary storage



Basic Components of a Computer

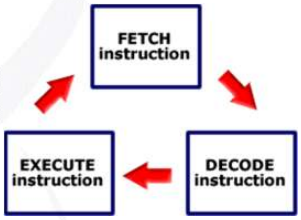
- **System bus**
  - The communications path that connects the CPU to the other components and peripherals in the system
    - Conducting channels that have high or low voltages
    - More simply, a set of 'wires' that carry bits
  - Consists of 3 parts
    - **Data** bus
    - **Address** bus
    - **Control** bus
  - In general, all components in a system share the same bus
  - The control unit in the processor coordinates communications





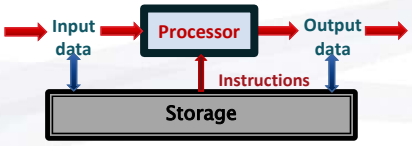
How a Processor works *(recap Module 1)*

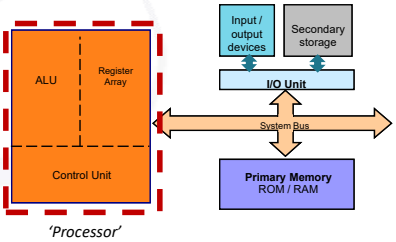
- A processor needs instructions to tell it what operations to perform on what data
- Instructions are stored in memory
- The microprocessor:
  - *fetches* an instruction from memory
  - *decodes* it, and
  - *executes* the specified operation
- Sequence of *fetch*, *decode* and *execute* continues indefinitely
  - Until reach an instruction to stop or powered off



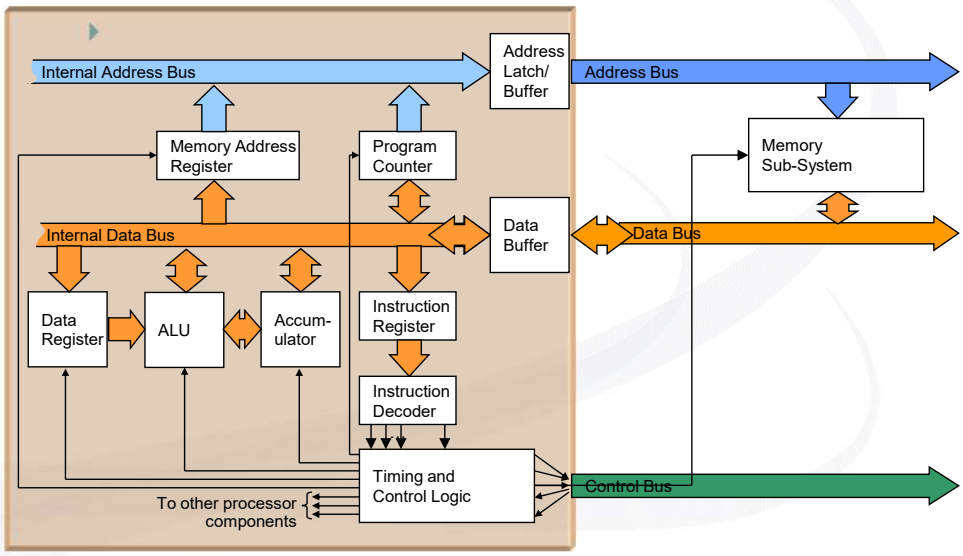
How a Computer System Works

- The processor uses the system bus to fetch the binary instructions and data from memory
- It uses registers for temporary storage of data and results
- It performs the computing operations on the data in the ALU
- And it sends out the results in binary, using the same bus lines, to the output unit or back into memory

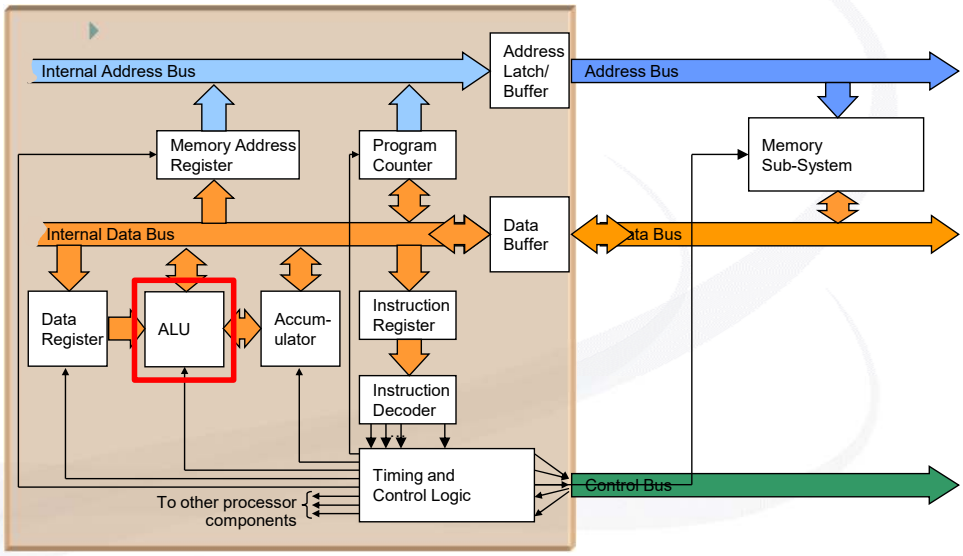




Simplified Diagram of a Processor

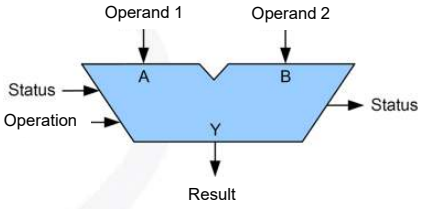


Simplified Diagram of a Processor



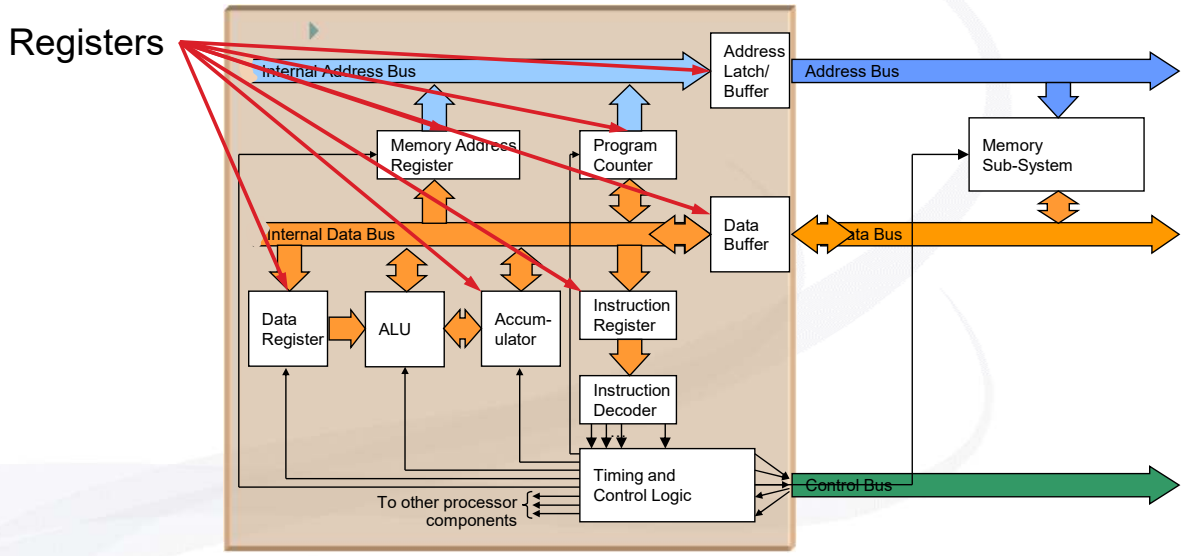
The Arithmetic Logic Unit (ALU)

- Electronic hardware designed to perform operations
  - Arithmetic operations : add, subtract, multiply, etc.
  - Logic operations : AND, OR, NOT, etc.
  - Binary data operations : Shift left/right, rotate, etc.



- Data to operate on (called operands) come from connected registers
  - Can have one or two operands, depending on operation
- What operation to perform is determined by signals from the Control unit
  - Depends on current instruction
- The result is stored in a register (most commonly the *accumulator*)
- *More details in Module 4*

Simplified Diagram of a Processor



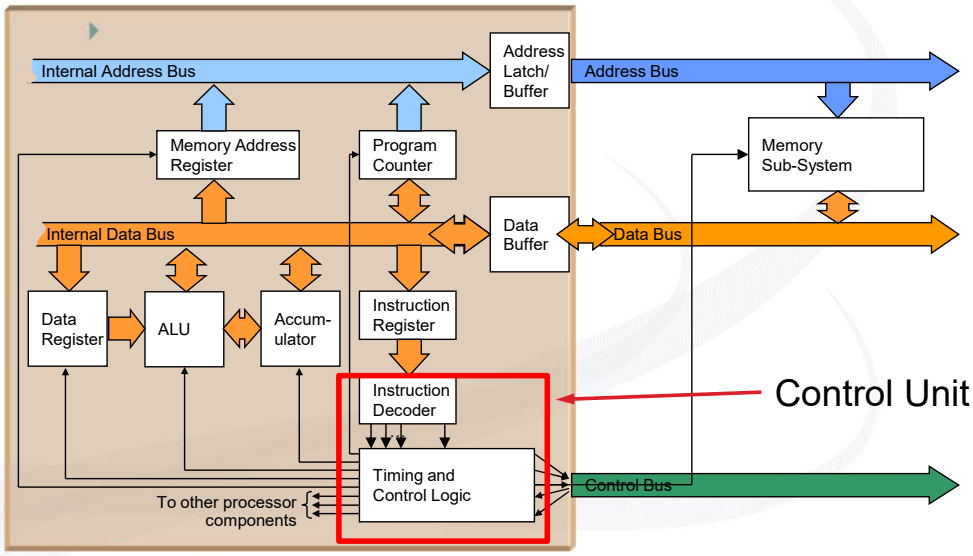
Registers

- A group of memory circuits used to store binary data

1	0	0	1	0	1	1	0

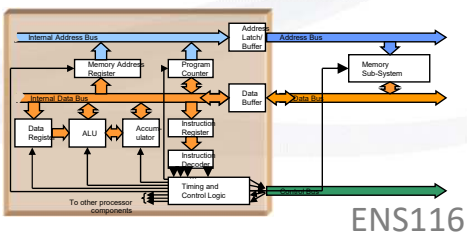
- High speed memory within the processor chip
- General purpose registers
  - Have no specific function except to store data
- Special function registers have specific roles to play in processor operation
  - **Instruction Register:** holds the *current instruction* to be decoded and executed
  - **Program Counter (PC) :** holds address of *next* instruction to be fetched
  - **Accumulator:** Data register that holds output of ALU
  - **Memory address register (MAR):** Holds address of operand to be fetched from memory
  - **Status register:** Holds logical status 'flags' (true/false)

Simplified Diagram of a Processor



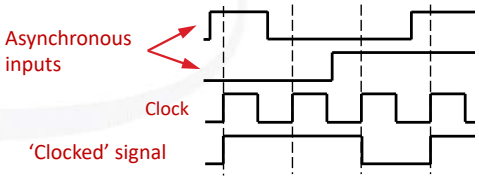
Control Unit

- Digital circuitry that coordinates all activities in the processor
  - fetches and decodes instructions
  - coordinates the movement of data within the system
  - provides signals to other parts of the system to 'tell' them what to do and when
    - Includes timing circuitry such as clock signals
  - Receives status signals various parts of system

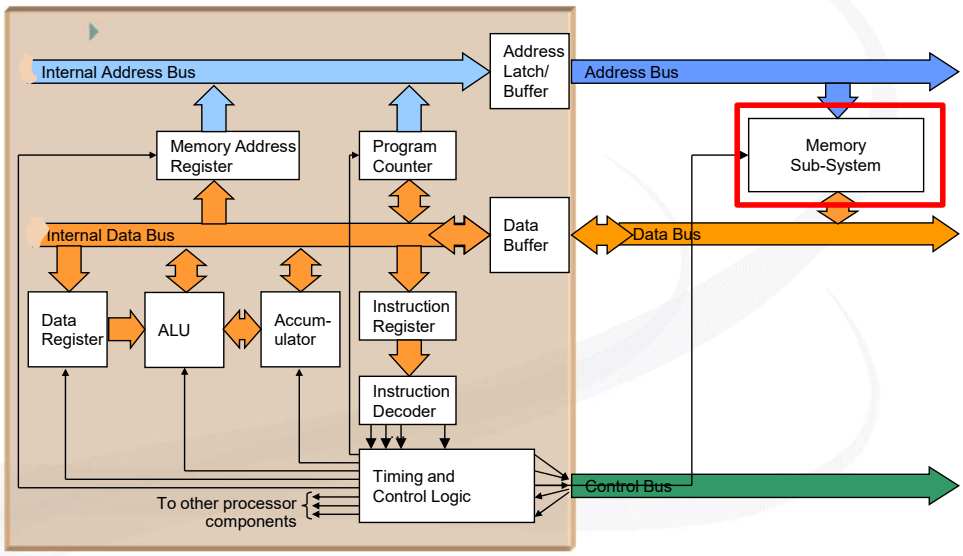


Clock Signals

- In order to coordinate the various components, the control unit synchronises their actions to clock signals
  - The 'drumbeat to which all the components march'
- An oscillator provides a precise timing signal to the processor
  - E.g. crystal oscillators
- Modern processors have base clock speeds in the GHz range
  - 1 GHz = 10^9 cycles pers second
  - These clock speeds may be *divided down* to slower clock signals for various components

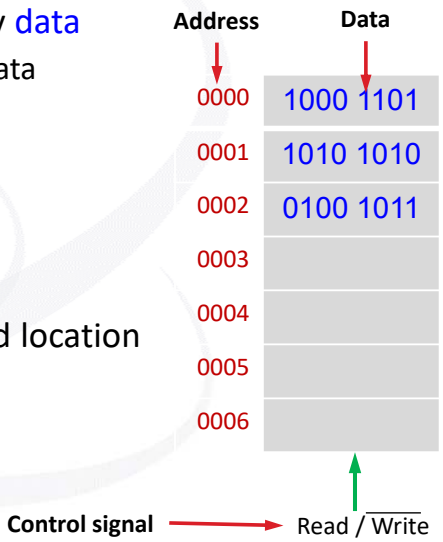


Simplified Diagram of a Processor

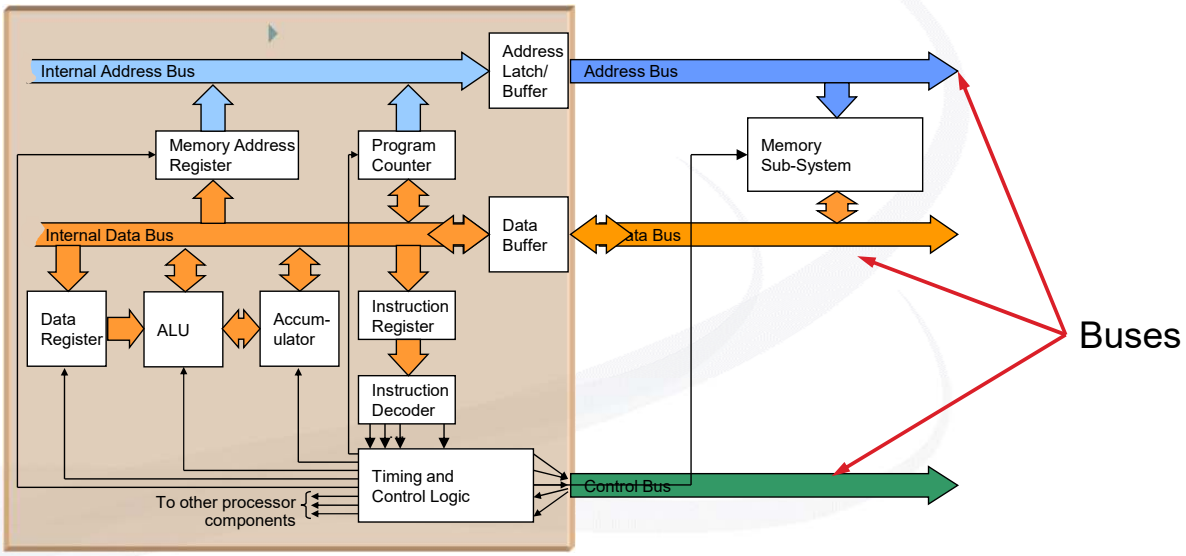


Memory

- Multiple storage locations, each containing binary data
  - Generally, each location contains 8 bits (1 byte) of data
  - Could contain multiple bytes (e.g. 16 bits, 32 bits)
- Each location has an address
  - Used to specify location to use
  - Has to be set first so 1 location is selected
- Data can be read from or written into the selected location
  - Depends on the control signal
  - *Memory operation covered in more detail in Module 6*

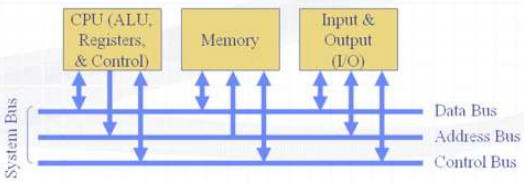


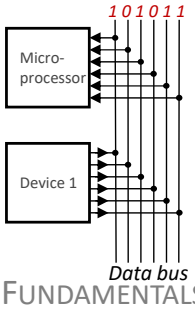
Simplified Diagram of a Processor



Buses

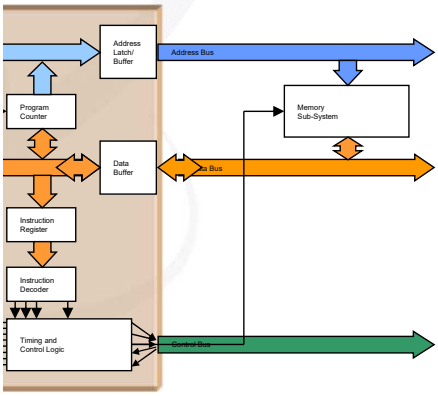
- A bus is essentially a collection of conducting channels or ‘wires’
- Each channel or line has can be at a ‘high’ voltage (1) or ‘low’ voltage (0)
  - So 1 line can ‘carry’ 1 bit of information
  - Multiple lines are needed to carry multiple bits (8, 16, 32, etc.)
- Buses are used for communication between devices in a computer system
- The main processor system bus consists of
  - the address bus
  - the data bus
  - the control bus





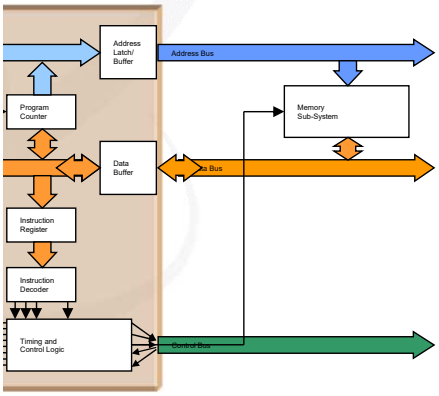
Address bus

- Used to select locations within the processors addressable space for reading or writing data
- The wider (more bits) the address bus, the greater the addressable space
  - 2^N addresses from an N-bit address bus
    - E.g. 8-bit address bus, $2^8 = 256$ addresses
- The address bus is unidirectional
  - From the processor to memory or device
  - No information is read from it



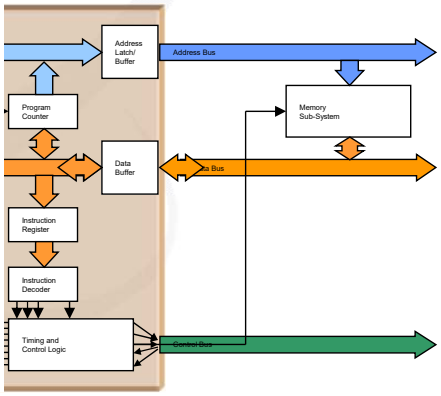
Data bus

- Used to transfer data to and from memory (or other device)
- All information in the system is transferred through this bus
  - data, program instructions, operand addresses, etc.
- Width of the data bus depends on processor
  - Normally the number of bits processor can process at one time
- The data bus is bidirectional
  - READ operation: data from *memory location specified on address bus* → processor
  - WRITE operation: data from processor → *memory location specified on address bus*



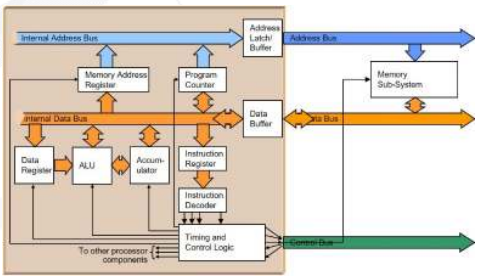
Control bus

- All the timing, control and power signals needed to drive the system and synchronise operations with other components in the system
- Control lines may be
  - outputs from the processor
  - inputs from external devices
- Common important signals:
  - clock signals
    - To synchronise actions of different components
  - the $R/\overline{W}$ signal
    - generated by processor to specify *Read* (when *High*) or $\overline{Write}$ (when *Low*) operation to addressed location
      - **Note:** the $\overline{\ }$ denotes what happens when signal is *Low* i.e. *0*



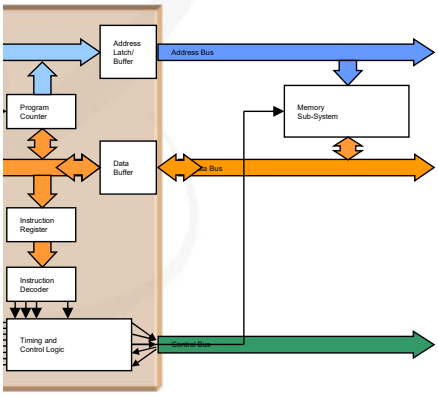
Internal vs External buses

- The *internal* address and data buses connect the processor components within the processor chip
  - Registers, ALU, etc.
- The *external* data bus connects the processor chip to external components
  - Memory, I/O devices
- Internal bus transfers are much faster because:
  - Transfers are within chip
    - Shorter distance, less interference
  - On chip components are faster
    - E.g. registers have a much quicker response time compared to external memory



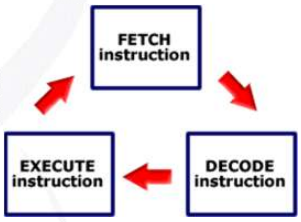
Address and data buffers

- Registers used to hold information to be written to or read from buses
- Sit in between internal and external buses – between processor and the rest
- **Address latch / buffer**
  - Address received from PC or MAR via internal bus
  - Holds (*'latches'*) the binary address written to the external address bus even if PC or MAR contents change
- **Data buffer**
  - Temporary storage for data read from external data bus or to be written via external bus to memory or I/O
  - Control unit sends signals to send data to/from various internal locations via internal data bus
    - Instruction register, accumulator, ALU, MAR, etc.



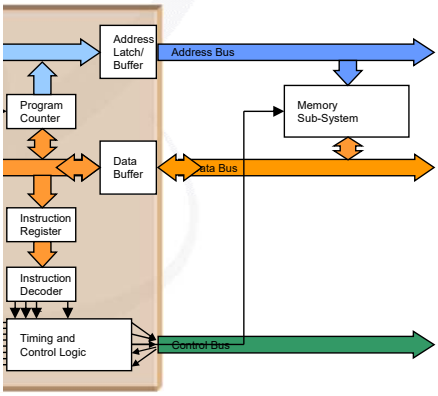
How a Processor works *(again)*

- A processor needs instructions to tell it what operations to perform on what data
- Instructions are stored in memory
- The microprocessor:
  - *fetches* an instruction from memory
  - *decodes* it, and
  - *executes* the specified operation
- Sequence of *fetch*, *decode* and *execute* continues indefinitely
  - Until reach an instruction to stop or powered off



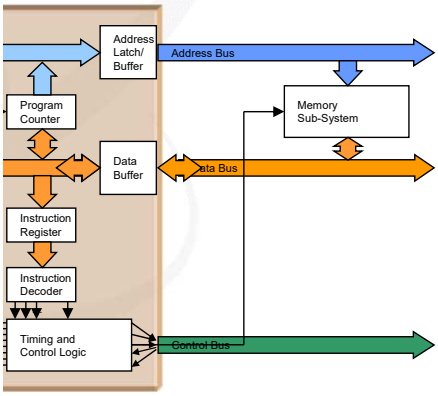
Instruction Fetch

1. Address of next instruction that is in the Program Counter (PC) register is put on the address bus
2. Memory location selected is read
   - Control signal Read sent to memory
3. Data in selected memory location is transferred via the data bus to the Instruction Register (IR)
4. Program Counter is incremented by 1
   - Ready to fetch the next instruction
   - In symbols: PC ← PC + 1



Instruction Decode

- Instruction Register holds the opcode that has been fetched
- Opcode – stands for *operation code*
  - binary code that tells the control unit what operation to perform
- The opcode is decoded by the Control Unit
  - Based on the opcode sequence of 1's and 0's the Control Unit generates a sequence of control signals
  - This is to complete the execution of the instruction



Instruction Set

- A processor can ONLY execute the set of instructions that has been built to understand
  - This is *the set of operations a processor can perform*
  - Different for every processor family
- Each instruction (or variation of it) has a unique opcode
- Each instruction requires specific *series of simple steps* to complete the required operation
  - Normally done by a built-in *microprogram* (or *microcode*) in the Control Unit
    - Generates the controls signals for the instruction in the right order and at the right time
  - In the *Execution* phase, each opcode results in a separate microprogram being run

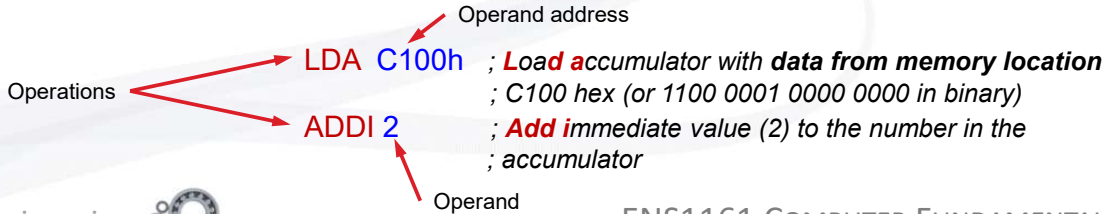
Instruction Set

- Sample instruction set - for 'generic' processor

Instruction description	Mnemonic	Instruction Code (Hex)	Instruction Code (Binary)
Load Accumulator Immediate	LDI	B5	1011 0101
Load Accumulator data from memory	LDA	B6	1011 0110
Store Accumulator to memory	STA	B7	1011 0111
…	…	..	
Add Accumulator Immediate (8-bit)	ADDI	BA	1011 1010
Add Accumulator with data from memory (8-bit)	ADD	BB	1011 1011
…	…	…	…
Unconditional Jump (absolute address)	JMP	D0	1101 0000
Jump if Zero - Z flag set (relative)	JZ	D1	1101 0001
Jump if Not Zero - Z flag clear (relative)	JNZ	D2	1101 0010
Jump if Carry - C flag set (relative)	JC	D3	1101 0011
Jump if Not Carry - C flag clear (relative)	JNC	D4	1101 0100
Jump if oVerflow - V flag set (relative)	JV	D5	1101 0101
Jump if Not oVerflow - V flag clear (relative)	JNV	D6	1101 0110

What is a program?

- A program is a series of instructions
  - stored in the memory sub-system
  - fetched and executed sequentially
- Each instruction consists of two parts
  - an *operation*
  - an *operand* or *operand address*
    - data / location of the data on which to perform the operation



Assembly language

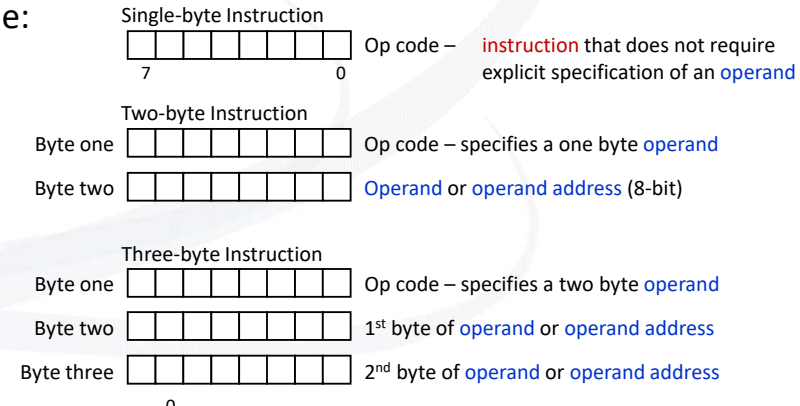
- A program is a series of instructions in binary that the processor can read and interpret (*decode*)
- Difficult to write programs in *1*s and *0*s, so use *assembly language*
- **Assembly language**
  - programs use 3 or 4 character mnemonic English abbreviations
  - mnemonics correspond to the binary opcodes that the processor understands
    - E.g. **LDA** = 1011 0110 (*B6* in hexadecimal), **ADDI** = 1011 1010 (*BA*h)
  - translated to binary using an assembler program *(more in Module 5)*

```
LDA  C100h   ; Load accumulator with data from memory location
             ; C100 hex (or 1100 0001 0000 0000 in binary)
ADDI 2       ; Add immediate value (2) to the number in the
             ; accumulator
```

1011 0110
1100 0001
0000 0000
1011 1010
0000 0010

Instruction Format

- The instruction format will depend on the processor
- Even in within one processor, different operations may require different instruction formats
- An 8-bit processor example:



Fetching the operand(s)

- If there are operands/operand addresses, they will need to be fetched from memory as part of the execution phase
  - This will be based on the result of the instruction decode
  - Control unit will then know how many bytes to fetch
- Process:
  1. Program counter value will be put on the address bus
  2. First byte of operand will be read from memory (via data bus) and stored at required location
  3. Program counter will be incremented
     - Ready pointing to next memory location
  4. Steps 1 – 3 repeated for as many bytes required

Memory address (Hex)	Memory contents (Hex)	
C000	B6	Opcode
C001	*C1*	2-byte Operand address
C002	*00*	

Sample Program

- *Hypothetical* generic 8-bit processor (*big-endian*)
- Program adds two 8-bit binary numbers (*X* and *Y*) stored in memory at locations C100h and C101h, the sum is stored in C102h

Memory address (Hex)	Memory word (Hex)	Mnemonic	Description
C000	B6	LDA	; LOAD accumulator
C001	*C1*		; *Address of*
C002	*00*		; *operand X*
C003	BB	ADD	; ADD *Y* to contents of accumulator
C004	*C1*		; *Address of*
C005	*01*		; *operand Y*
C006	B7	STA	; STORE the contents of accumulator
C007	*C1*		; *Address where the contents of*
C008	*02*		; *accumulator will be stored*

Processor Operation

- *CPU Animation* on Blackboard shows the following microcode steps that the processor carries out to run the program shown

Sample Program Execution

- To begin, the PC must be set to point to the address of the first instruction (C000h)
  1. Fetch opcode;
     - Control Unit (CU) places PC = C000h onto address bus
     - sets R/$\overline{W}$ = 1 (i.e. *Read* signal)
     - the opcode (B6h, *LDA*) can then be read from the data bus into IR
       [C000h] → IR
  2. Increment PC
     - PC +1 → PC ready to fetch the next part of the instruction PC = C001h
  3. Decode opcode
     - the instruction decoder decodes the op-code (B6h) which is code for LDA
     - signals the CU to begin generating the sequence of control signals to **execute** the LDA instruction (microprogram)

Memory address (Hex)	Memory contents (Hex)
C000	B6
C001	*C1*
C002	*00*
C003	BB
C004	*C1*
C005	*01*
C006	B7
C007	*C1*
C008	*02*

Sample Program Execution *(cont.)*

4. Fetch operand address (1st byte)
   - CU places PC = C001h on the address bus
   - set R/$\overline{W}$ = 1
   - causes C1h to appear on the data bus, loaded into the high byte of MAR — [C001h] → MAR_{high}
5. Increment PC — PC = C002h
6. Fetch operand address (2nd byte) — [C002h] → MAR_{low}
   - MAR now contains the full 16-bit operand address C100h
7. Increment PC — PC = C003h
8. Perform LDA (*Load Accumulator*) function
   - the operand address from MAR is placed on address bus
   - R/$\overline{W}$ is set high
   - operand *X* appears on the data bus
   - Operand loaded into the accumulator — [C100h] → A

Memory address (Hex)	Memory contents (Hex)
C000	B6
C001	*C1*
C002	*00*
C003	BB
C004	*C1*
C005	*01*
C006	B7
C007	*C1*
C008	*02*
…..	
C100	*X*
C101	*Y*
C102	*Sum*

Sample Program Execution *(cont.)*

9. Fetch next op-code
   - CU places PC = C003h on the address bus
   - sets R/$\overline{W}$ = 1
   - *BB*h (opcode for ADD) appears on the data bus
   - loaded into IR — [C003h] → IR
10. Increment PC — PC = C004h
11. Decode op-code
    - the instruction decoder decodes it as the ADD instruction
    - signals the CU to begin execution
12. Fetch operand address (1st byte) — [C004h] → MAR_{high}
13. Increment PC — PC = C005h
14. Fetch operand address (high byte) — [C005h] → MAR_{low}
    - MAR now contains C101h
15. Increment PC — PC = C006h

Memory address (Hex)	Memory contents (Hex)
C000	B6
C001	*C1*
C002	*00*
C003	BB
C004	*C1*
C005	*01*
C006	B7
C007	*C1*
C008	*02*
…..	
C100	*X*
C101	*Y*
C102	*Sum*

Sample Program Execution *(cont.)*

16. Perform ADD
    - the operand address from MAR is placed on the address bus
    - R/$\overline{W}$ set high
    - operand *Y* appears on the data bus
    - *Y* is loaded into the data buffer/register,
    - the ALU is then signalled to ADD this value to A
    - place the result into A — [C101h] + A → A
17. Fetch next op-code
    - CU places PC = C006h on the address bus
    - sets R/$\overline{W}$ = 1
    - *B7*h (opcode for STA) appears on the data bus
    - loaded into IR — [C006h] → IR
18. Increment PC — PC = C007h

Memory address (Hex)	Memory contents (Hex)
C000	B6
C001	*C1*
C002	*00*
C003	BB
C004	*C1*
C005	*01*
C006	B7
C007	*C1*
C008	*02*
…..	
C100	*X*
C101	*Y*
C102	*Sum*

Sample Program Execution *(cont.)*

19. Decode op-code (*B7*h = STA)
20. Start executing STA (*Store Accumulator*)
21. Fetch operand address (1st byte) — [C007h] → MAR_{high}
22. Increment PC — PC = C008h
23. Fetch operand address (2nd byte) — [C005h] → MAR_{low}
    - MAR now contains C102h
23. Increment PC — PC = C009h
24. Perform Store
    - operand address from MAR is placed onto the address bus
    - the CU then places A onto the data bus
    - clears R/$\overline{W}$ to 0 (i.e. *Write* signal)
    - A gets written to memory location C102h — A → [C102h]

Memory address (Hex)	Memory contents (Hex)
C000	B6
C001	*C1*
C002	*00*
C003	BB
C004	*C1*
C005	*01*
C006	B7
C007	*C1*
C008	*02*
…..	
C100	*X*
C101	*Y*
C102	*Sum*

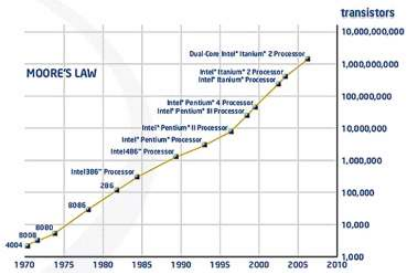
Key points regarding process

- Each instruction involves a number of steps
  - Includes fetching and decoding the instruction
  - Number of microcode steps depends on the instruction and data required
  - Each step takes time
  - Memory fetches are comparatively slow
    - *More in Module 6*
- Even something as basic as adding 2 numbers and storing the answer:
  - Took 3 instructions
  - Broken down into 24 steps
  - Each step will require a number of control signals in particular sequence

Speed of processing

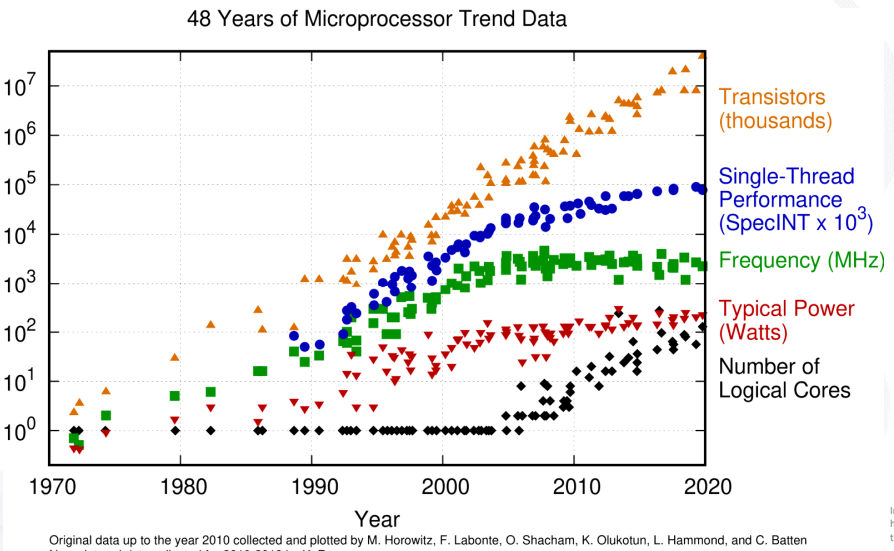
- Each microcode step takes 1 or more clock cycles
- One instruction takes a number of clock cycles to complete
  - The time taken depends on the complexity of the instruction
- The processor does thousand or millions of these steps per second, so the overall process is very fast
  - Partly depends on clock speed
  - Also depends on other factors, such as:
    - Memory access speed (can be a bottleneck)
      - *Explored more in Module 6*
    - Width of data bus / word length
      - More data can be transferred / processed at one go

General trends *(recap module 1)*



- Computers have been shrinking in physical size
  - Due to advances in electronics
- Computing power has grown exponentially
  - The speed and amount of data that can be processed
- The cost of computers has (generally) dropped
- The number computers / computing devices has grown exponentially
- The amount of data being produced has grown exponentially
- The cost of storage has dropped dramatically
- Connectivity (ability to communicate) between computers has become a key factor
- All of the above have made computers disrupt how we work and play

Processor trends



Karl Rupp, 2020

Image: Karl Rupp, 2020
https://github.com/karlrupp/microprocessor-trend-data/

Factors affecting increasing performance

- Increasing clock speed has certain limits
  - Faster switching results in more heat being generated
  - Other components such as memory can't keep up
- Trend over last 15 years – multi-core processors
  - Each 'core' is essentially a processor on its own
  - Essentially multiple processors on one chip
    - E.g. dual core, quad core, octa-core
  - More complexity in sharing of data and code
- Cramming more transistors in smaller space also creates more heat
  - Most modern processors require heatsinks and other cooling mechanisms

Heatsink and fan on processor

Image: Fir0002/Flagstaffotos, 2006
https://en.wikipedia.org/wiki/Computer_cooling#/media/File:AMD_heatsink_and_fan.jpg

Summary

- Basic components of a computer
  - **The Processor (ALU, registers, control unit):**
    - Reads instructions from memory (*fetches*)
    - Instructions in binary are then interpreted (*decodes*)
      - Each processor has a predefined set of instructions it understands (*instruction set*)
    - Performs (*executes*) the tasks specified in the instruction
      - Each instruction is broken into a series of steps (*microcode*) orchestrated by the Control Unit
    - Communicates with all peripherals (memory and I/O) using the system bus
    - Controls the timing of information flow

Summary

- **Memory**
  - Stores instructions and data in binary format
  - Provides these instructions and data to the processor on request
  - Stores results data from the processor
- **The Input Unit:**
  - Receives data and/or instructions from external sources and passes them to the processor or to memory
- **The Output Unit**
  - Passes results data from the processor or from memory to external devices or peripherals

Summary

- **The System Bus**
  - Channel that transports **addresses**, instruction or **data** bits, as well as **control** signals
  - Externally between the processor, the memory and the I/O units
  - Internally between processor components
- **Program:**
  - A list of instructions that is (normally sequentially) fetched, decoded and executed by the processor.
  - The instructions must be in the form understood by the processor (instruction set).

Unit Learning Outcomes

On completion of this unit, students should be able to:

- **Describe the fundamental architecture and operating principles of a computer system.**
- Interpret computer system specifications and standards and how they relate to system function.
- Compare different types of components and subsystems and their relative impacts on system function and performance.
- Make recommendations on the suitability of computer systems and components for a given function.
- Explain the interconnection between the software and hardware components of computer systems, and the processes involved in making them work together.

Next Module

Module	Topic
1	Introduction and overview
2	Basic Computer Architecture and Principles of Operation
3	Data and instruction formats
4	Basic computation in computers
5	Programming languages and tools
6	Memory devices and storage systems

Module	Topic
7	I/O devices and interfacing
8	I/O modes and BIOS
9	Operating Systems
10	Networks and the Internet
11	Embedded Systems and Cloud Computing
12	Review & Revision