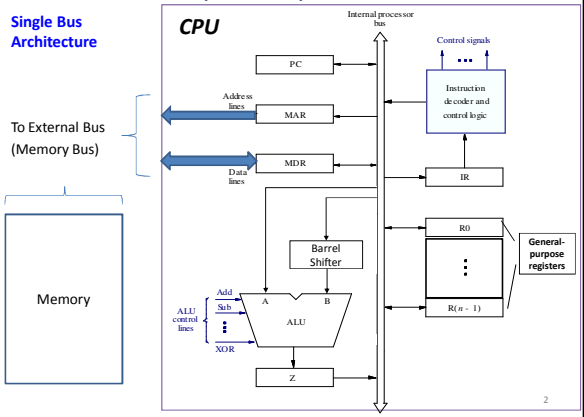
UECS1013 Introduction to
Computer Organisation and Architecture

Components of a Computer

Computer Operations



Computer Operations

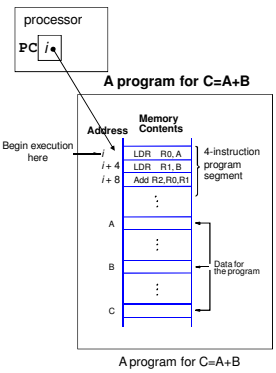
- **Components of the processor :**
  1. **Arithmetic and Logical Unit (ALU)**
     - Made up of circuits that perform the arithmetic and logical execution within the processor.
     - It has no internal storage accessible to a programmer.
  2. **Control Unit (CU)**
     - It contains circuits that direct and coordinate proper sequence, interpret each instruction and apply the proper signals to the ALU and registers.
  3. **Registers**
     - Registers are high speed temporary data storage area within the processor to support execution activities.
     - Both instructions or data can be stored in registers for processing by the ALU.
  4. **Bus**
     - The bus is the interconnection lines used to transfer data between the various components. The **data** and **address lines** of the external memory bus are connected to the internal processor bus via MDR and MAR respectively. The **control lines** of the memory bus are connected to the Instruction decoder and control logic block.

Computer Operations

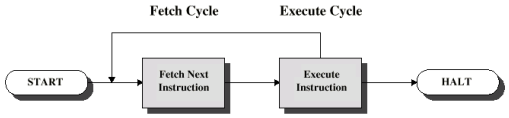
- **Special Registers**
  - The **Instruction Register (IR)** holds the instruction that is currently being executed. Its output is available to the control unit, which generates the timing signals that control the various processing elements involved in executing the instruction.
  - The **Program Counter (PC)** contains the memory address of the next instruction to be fetched and executed.
  - The **Memory Address Register (MAR)** holds the address of the main memory location to be accessed.
  - The **Memory Data Register (MDR)** contains the data to be written into or read out of the addressed main memory location.

Computer Operation

- ***To perform a task***, a program consisting of a list of instructions is stored in the memory.
  - Data to be used as operands are also stored in the memory.
- ***To execute a program***, the processor fetches one instruction at a time from the memory, and performs the operations.
- *Notes*: The details of these operations are hidden to an assembly language programmer
- How is an instruction executed?



Instruction Cycle



- **Fetch Cycle:**
  - Transfer an instruction from memory through the bus;
  - interpret and decode the instruction
  - Compute the default location of the next instruction
- **Execute Cycle:**
  - carry out the instruction

Fetch Cycle

- In the Fetch cycle, the instruction is fetched from the memory location pointed to by the *Program Counter* (PC) into the *instruction register* (IR). The processor keeps track of the address of the memory location containing the next instruction to be fetched using the PC.

IR ← [PC] can be further broken down as:

MAR ← PC
MDR ← [MAR]
IR ← MDR

- Assuming a byte-addressable machine and each instruction comprises 4 bytes, the Program Counter (PC) is updated as follows.

PC ← PC + 4

This may be different for different processors. After fetching an instruction, the PC is updated to point to the next instruction.

7

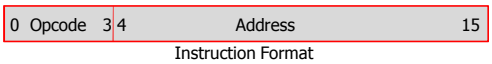
Execute Cycle

- In the **Execute Cycle**, the instruction in IR is examined to determine which operation is to be performed.
- Most of the operations needed to execute an instruction can be carried out by performing the following functions in some specified sequence.
  - ✓ **Processor-memory –** data transfer between CPU and Main Memory
  - ✓ **Processor I/O** – data transfer between CPU and I/O Module
  - ✓ **Data Processing** – some arithmetic or logical operation on data
  - ✓ **Control** – Alteration of sequence of instructions, e.g. jump/branch
  - ✓ Combination of the above.
- After the execution of the current instruction is completed, a new instruction fetch may be started

8

Example of Program Execution

- Assume a particular computer system with the following instruction format:



Instruction Format

Partial List of Opcodes
0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

Internal CPU Register
PC = Address of next instruction.
IR = Instruction being executed
AC = Temporary storage

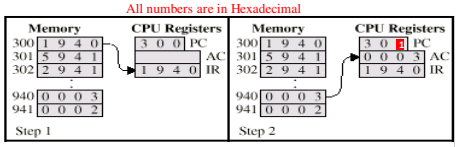
Consider the following machine code (shown in hexadecimal) loaded to memory block 0x300
1940
5941
2941
Assume that current value of the PC is 0x300 and the system memory is half-word addressable (different from ARMS which is byte-addressable).

9

Review Question

All numbers are in Hexadecimal

Instruction 1:
1940
AC ← [940]



First Instruction:
Fetch Cycle:
1. The PC contains 300 the address of the first instruction.
2. This instruction (the value 1940 in hex) is loaded into the instruction register, IR.
3. The PC is incremented at the last step (updated value is shown only in the right diagram). Note that this process would involve the use of a memory address register (MAR) and a memory buffer register (MBR). For simplicity, these intermediate registers are ignored.

Execute Cycle:
1. Decoding and executing the Instruction: The first 4 bits (first hex digit –1) in the IR indicate that the AC is to be loaded. The remaining 12 bits (three hex digit) specify the address (940) from which data are to be loaded

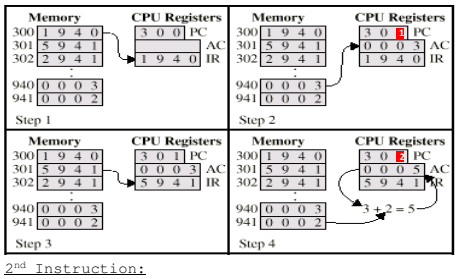
10

Review Question

All numbers are in Hexadecimal

Instruction 1:
1940
AC ← [940]

Instruction 2:
5941
AC ←AC + [941]



2nd Instruction:
- The next instruction (5941) is fetched from location 301 and the PC is incremented
- The old contents of the AC and the contents of location 941 are added and the result is stored in the AC.

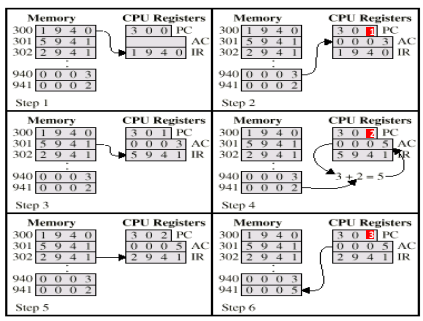
Review Question

All numbers are in Hexadecimal

Instruction 1:
1940
AC ← [940]

Instruction 2:
5941
AC ←AC + [941]

Instruction 3:
2941
[941] ← AC



Instruction 3
- The next instruction (2941) is fetched from location 302 and the PC is incremented.
- The contents of the AC are stored in location 941.

12

Extended Instruction Cycle

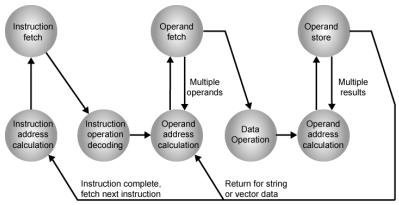
- Some CISC processor may contain complicated instructions involving more than one memory reference.
- For example, PDP-11 processor have the following instruction:

  ADD B, A

  Which performs the following operation [A] ← [A] + [B]
- For such processor, a ***single*** instruction cycle contains the following steps:
  1. Fetch + Decode the instruction
  2. Read the contents of memory location A into the processor
  3. Read the contents of memory location B into the processor (requires at least two registers for storing memory values)
  4. Add the two values
  5. Write the result from the processor to memory location A

13

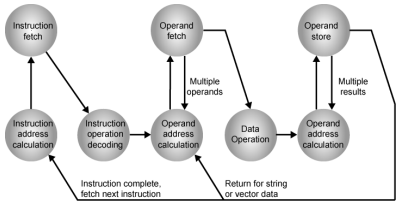
Extended Instruction Cycle



- **Instruction Address Calculation (IAD):**
  - ✓ Determine the address of the next instruction.
  - ✓ Without flow control: Add PC with a fixed number to point to the next address
  - ✓ With flow control: Update PC with the effective address of the destination branch instruction
- **Instruction Fetch (IF)**
  - ✓ Read instruction from its memory location into the processor

14

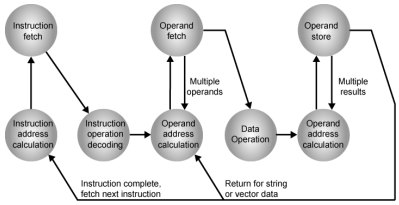
Instruction Cycle State Diagram



- **Instruction Operation Decoding (IOD):**
  - ✓ Analyze instruction to determine type of operation to be performed and operand(s) to be used
- **Operand Address Calculation (OAC):**
  - ✓ If the operation involves reference to an operand in the memory or available via I/O, then determine the address of the operand
- **Operand Fetch (OF):**
  - ✓ Fetch the operand from the memory or read it from I/O

15

Instruction Cycle State Diagram



- ✓ OAD and OF may be repeated if the instruction contains multiple references to the memory (only applicable to CISC)
- **Data Operation (DO):**
  - ✓ Perform the operation indicated in the instruction
- **Operand Store (OS):**
  - ✓ Write the result into the memory or out to I/O

16

INTERRUPTS

Interrupts

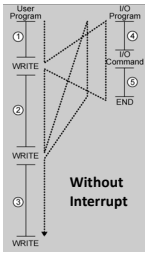
- All computers provide a mechanism by which other modules (I/O, memory) may interrupt the normal processing of the CPU
- Different types of Interrupts:

Type	Description
Program	Generated by some condition when executing an instruction, e.g., arithmetic overflow, division by zero, illegal instruction, invalid memory access, etc.
Timer	Generated by a timer within a processor to allow the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller to signal normal completion of an operation or to signal a variety of error conditions
Hardware Failure	Generated by a hardware failure such as power failure or memory parity error

18

Increasing Processing Efficiency through Interrupts

- One of the main function of interrupt is to improve processing efficiency when accessing external devices which are much slower than the CPU.
- Avoid wasting large processing cycles to wait for slow devices to complete their tasks. For example, when transferring data to a printer using the original instruction scheme (without instruction cycle), the processor needs to remain idle while waiting for the printer to catch up → delay by hundreds/thousands of instruction cycles.



User Program

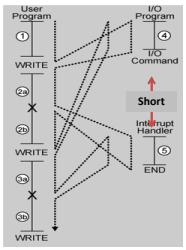
- ***Sequence 1, 2 and 3*** refer to the instructions not involving any I/O.
- **WRITE** calls to a program that is a system utility that will perform the I/O operations

I/O Program

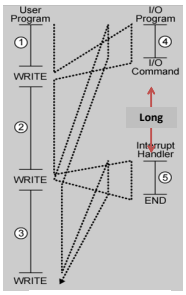
- ***Sequence 4*** prepares for the actual I/O operation (e.g. copy output data to a special buffer)
- ***I/O Command*** issues the actual I/O operation in the I/O Program, e.g., the actual printing
- ***Sequence 5*** completes the operation in the I/O Program, e.g., setting a flag to indicate the success or failure of the operation.

19

Program Timing: Short vs Long I/O Operations



Short I/O Operation

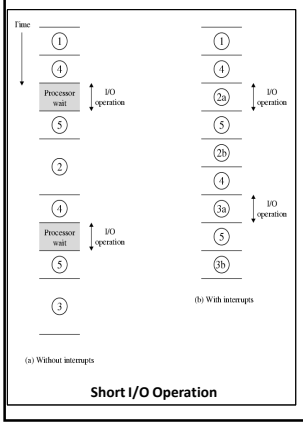


Long I/O Operations

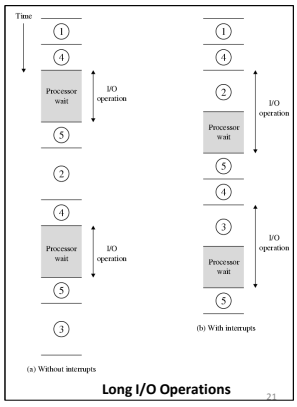
If the I/O operation is longer than the time to complete the (2), the second WRITE command needs to wait until the first WRITE command completes. However, there is still a gain of efficiency

20

Program Timing: Short vs Long I/O Operations



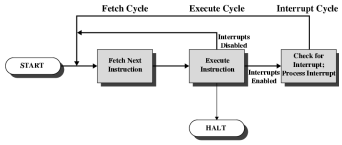
Short I/O Operation



Long I/O Operations

21

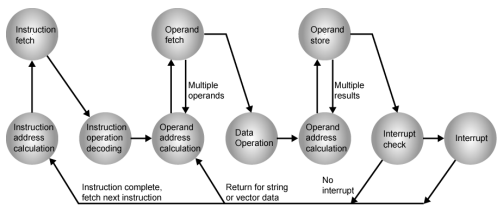
Interrupt Cycle



- To support interrupts, the ***Interrupt Cycle*** to the instruction cycle. During this cycle, the processor checks if any interrupt by checking the interrupt signal.
- If no interrupt, fetch next instruction
- If interrupt pending:
  - Suspend execution of current program
  - Save context
  - Set PC to start address of interrupt handler routine and process the interrupt. The interrupt handler determnes the nature of the interrupt and performs whatever actions that are needed. For example, it determines which I/O module generate the interrupt and may branch to another program to write more data out to that I/O module.
  - Restore context and continue interrupted program

22

Interrupt Cycle



Instruction Cycle State Diagram with Interrupts

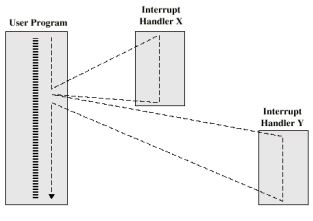
23

Multiple Interrupts

How should we handle multiple interrupts?

Disable interrupts

- Processor will ignore further interrupts whilst processing one interrupt. Interrupts remain pending and are checked after first interrupt has been processed
- Interrupts will be handled in sequence as they occur



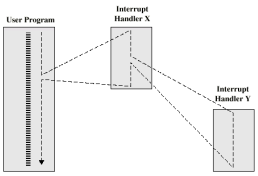
- Problem: does not take into consideration the relative priority or time-critical needs. For example, data may be lost in network communication if it is not serviced immediately as new data comes.

24

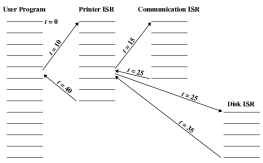
Multiple Interrupts

How should we handle multiple interrupts?

- Define priorities
  - Low priority interrupts can be interrupted by higher priority interrupts
  - When higher priority interrupt has been processed, processor returns to previous interrupt



Interrupt Y has a higher priority than Interrupt X

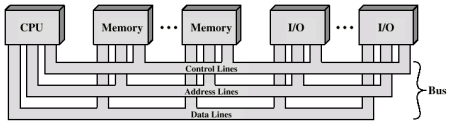


Communication ISR has a higher priority than Printer ISR

25

BUSES

What is a Bus?



- A communication pathway connecting two or more devices
- Grouped as a number of channels in one bus, e.g., 32 bit data bus is a 32 separate single bit channels. Each
- **System bus**: a bus that connects the major components (processor, memory, I/O).
- A computer system can deploy one or more system busses.
- A shared transmission medium. Therefore, only one device at a time can transmit.
- The lines in a bus can be categorized based on its function: *data, address* and *control* lines.

27

Control Lines

- Identify the source or destination of data, e.g., CPU needs to read an instruction (data) from a given location in memory
- Bus width determines maximum memory capacity of system, e.g., 8080 has 16 bit address bus giving 64k address space.
- Transmit both the *command* and *timing* information between modules.
- Typical control lines:
  - ✓ **Memory read/write**: Specify if the current memory operation is read/write
  - ✓ **I/O read/write**: Specify if current I/O access is read/write
  - ✓ **Transfer ACK**: indicates that data have been accepted from or placed on the bus
  - ✓ **Bus Request**: indicates that a module wants to gain control of the bus
  - ✓ **Bus grant**: indicates that a module has been granted control of the bus
  - ✓ **Interrupt request**: indicates that an interrupt is pending
  - ✓ **Interrupt ACK**: Acknowledges that the pending interrupt has been recognized
  - ✓ **Clock**: Used to synchronize operations
  - ✓ **Reset**: Used to initialize all modules

28

Address and Data Lines

Address Lines

- Used the designate the source or destination of data on the data bus. Also used to address I/O ports.
- Higher-order bits are used to select which I/O module.
- Lower-order bits are used to specify the memory location or I/O port within the module.
- For example: on a 8 bit address bus, one possible scheme is to use the most significant bit to select two different modules and the lower 7 bits to access the specific memory location or I/O ports where address 0xxxxxxx can be used to access the memory module with 128 words of memory, and address 1xxxxxxx can be used to access devices that are attached to the I/O module

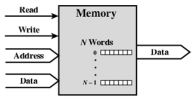
Data Lines

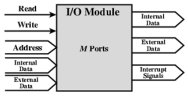
- Provide a path for moving data between system modules.
- The width of the bus determines how many bits can be transferred at a time → affects the overall system performance. For example, if the width of the data line is doubled, the number of memory access will be reduced by half.

29

Computer Modules

- All the units must be connected the bus
- Different type of connection for different type of unit

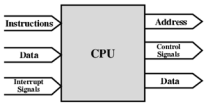




- Receives and sends **data**
- Receives **addresses**
- Receives **control** signals
  - Read
  - Write
  - Timing

- Similar to memory from computer's viewpoint but also act to interface the peripherals and computer
  - ✓ Receive **internal data** from computer and send the data to peripherals.
  - ✓ Receive **external data** from peripheral and send data to computer
- Receive **addresses** from computer
  - e.g. ***port*** number to identify peripheral
- Receive **control** signals from computer
- Send **control** signals to peripherals
  - e.g. spin disk
- Send **interrupt** signals (control)

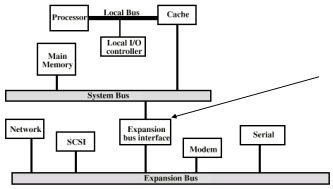
Computer Modules



- Reads **instruction** and **data**
- Writes out data (after processing)
- Sends **control** signals to other units
- Receives (& acts on) **interrupts**

Multiple Bus Architecture

- Many device connected to a single bus leads to the following problem:
  - ✓ Greater propagation delay: caused by greater bus length to accommodate more devices. When control of the bus passes from one device to another frequently, the propagation delay can noticeably affect performance.
  - ✓ Aggregated data transfer may approach the capacity of the bus.
- To overcome this issue, most computer systems use **multiple buses.**



- Buffers data transfers between the system bus and I/O controllers on the expansion bus
- Insulates the memory-to-processor traffic from I/O traffic

A traditional multi-bus architecture

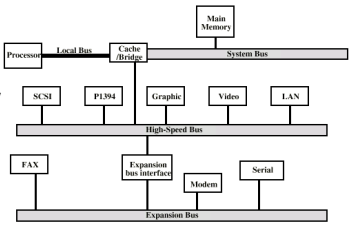
32

Multiple Bus Architecture

- Traditional multiple bus architecture starts to break down when higher and higher performance is seen in I/O devices.
- Solution: The **Messanine architecture** builds a high-speed bus that is closely integrated with the rest of the system, requiring only a **bridge** between the processor's bus and high-speed bus.
- Benefit: high-speed bus brings high-demand devices into close integration with the processor but at the same time is independent of the processor. Differences in processor and high-speed bus speeds are tolerated.

(Relatively) High speed Peripherals is supported by the high-speed bus

Low speed peripherals



33

Bus Design

- **Type**: *How the bus is organized: types of buses and their functions (address, instruction and data)*
  - Dedicated
  - Multiplexed
- **Arbitration Method:** *Determines which module takes control of the bus*
  - Distributed
  - Centralized
- **Timing:** *Refers to the way how events are coordinated on the bus.*
  - Synchronous
  - Asynchronous
- **Bus Width:** *How many bits can a bus transmit in a single transmission?*
  - Address
  - Data
- **Data Transfer Type**: *Types of data transfers supported by the bus*
  - Read
  - Write
  - Read-modify-write
  - Read-after-write
  - Block

34

Bus Types

- Dedicated
  - Separate data and address lines
- Multiplexed
  - **Time Multiplexing**: address and data information are transmitted over the same set of lines.
  - **Address Valid** or **Data Valid** control line to indicate whether address or the data is being transmitted
  - Advantage
    - Fewer lines
  - Disadvantages
    - More complex control
    - Ultimate performance

35

Bus Arbitration

- More than one module controlling the bus, e.g., CPU and **DMA controller**
- Only one module may control bus at one time → need arbitration.
- There are two types of arbitration:
  1. **Centralised** Arbitration
     - Single hardware device controlling bus access: **Bus Controller** or **Arbiter**
     - May be part of CPU or a separate module
  2. **Distributed** Arbitration
     - Each module may claim the bus
     - Control logic on all modules

36

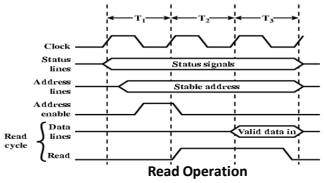
Synchronous Timing

- Occurrence of events on the bus is determined by clock signals
- Control Bus includes clock line. A clock is a regular sequence of alternating 0s and 1s of equal duration. A single 1-0 transmission is referred to as one ***clock cycle*** or ***bus cycle*** and is used to define a time slot.
- Usually a single cycle for an event
- Usually sync on the *rising* clock edge. Note that some delay will be observed at the signals after the triggering clock edge.

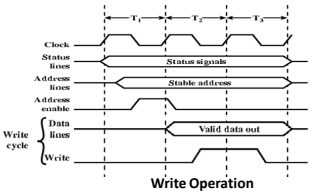
Notes: Signals with a bar means that it is asserted (enabled) when it has a value of zero. For example, $\overline{read}$ is enabled (signalling a read operation) only when it has a value of 0

37

Synchronous Timing



Read Operation



Write Operation

- T1:

  Source module, e.g., the processor, will generate the *address* (location) signals. Only after the address is stable, the processor issue the *address enable* signal.
- T2:

  Read operation: source module will assert the *read* signal.

  Write operation: source module places the *data* on the data lines and assert the *write* signal after the data lines have stabilized.
- T3:

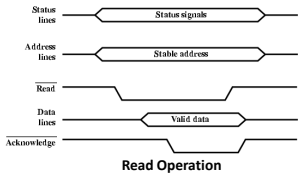
  Read operation: The memory supplies the *data* on the data line

  Write operation: The memory module copies the *data* in the data line.

38

Asynchronous Timing

- The occurrence of events on a bus follows the and depends on the occurrence of a previous event.

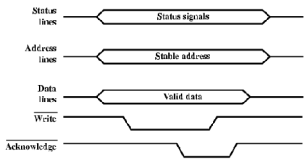


Read Operation

1. Processor issues the **address** and **status** signal.
2. After the address and status stabilizes, the processor issue the **read** command (indicating valid address and status signals)
3. The memory then decodes the address and responds by placing the **data** on the data line.
4. Once the data is ready, the memory module asserts the **acknowledge** signal to signal processor that the signal is available.
5. After the processor reads the data from the data lines, the processor drops the **read** signal.
6. After the read signal is dropped, the memory responds by dropping the data and **acknowledge** lines.
7. After the acknowledge signal is dropped, the processor will then stop issuing the **address** and **status** signal.

39

Asynchronous Timing



Write Operation

1. Processor issues the **status**, **data** and **address** signal.
2. After the address and status stabilizes, the processor issue the **write** command (indicating valid address and status signals)
3. The memory then decodes the address and responds by **copying** the data from the data lines.
4. The memory then asserts the **acknowledge** line.
5. The master drops the **write** signal
6. The memory drops the **acknowledge** signal.
7. The processor drop the **data**, **address** and **status** signals

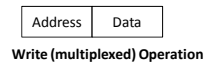
40

Synchronous vs Asynchronous Timing

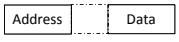
- Synchronous Timing
  - *Simpler* to implement and test
  - *Less flexible* than asynchronous timing. All devices are tied to a fixed clock rate. Therefore, the system cannot take advantage of advances in device performance.
- Asynchronous Timing
  - May deliver a better performance when *a mixture of slow and fast devices*, using older and newer technology share the bus at the cost of design complexity.

41

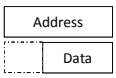
Data Transfer Type



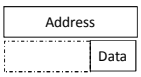
Write (multiplexed) Operation



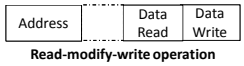
Read (multiplexed) Operation



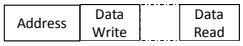
Write(non-multiplexed) Operation



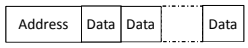
Read (non-multiplexed) Operation



Read-modify-write operation
(same address, to protect shared memory)



Read-after-write operation
(same address, for checking purposes)



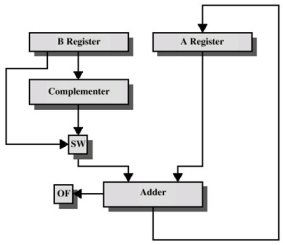
Block data Transfer

42

ARITHMETIC LOGICAL UNIT (ALU)

Addition & Subtraction

- Normal binary addition
- Monitor sign bit for overflow
- For subtraction, take the twos compliment of substrahend and add to minuend
  - i.e. a - b = a + (-b)
- So we only need addition and complement circuits



OF = overflow bit
SW = Switch (select addition or subtraction)

44

Unsigned Multiplication

- Complex
- Strategy:
  - Work out partial product for each digit
  - Take care with place value (column)
  - Add partial products

```
  1011   Multiplicand (11 in decimal)
 x1101   Multiplier    (13 dec)
  1011   Partial products
 0000    Note: if multiplier bit is 1 copy
1011           multiplicand (place value)
+1011          otherwise zero
10001111 Product (143 dec)
```

Note: need double length result

45

Unsigned Multiplication

```
  1011
 x1101
  1011  (1)
 0000   (2)
1011    (3)
+1011   (4)
10001111
```

In practice, the addition of (1) to (4) is done incrementally since the adder circuit takes only two inputs at a time

```
  1011
 x1101
 0000
+1011   (1)
 1011
+0000   (2)
 01011
+1011   (3)
 110111
+1011   (4)
10001111
```

46

Unsigned Multiplication

```
  1011
 x1101
 0000
+1011   (1)
 1011
+0000   (2)
 01011
+1011   (3)
 110111
+1011   (4)
10001111
```

Initialize
C 0 A 0000 Q

Add and Shift (Multiply with 1 1 0 1)
+ 1011
C 0 A 1011 Q — Add (1)
C 0 A 101 Q 1 — Shift

Add and Shift (Multiply with 1 1 0 1)
+ 0000
C 0 A 0101 Q 1 — Add (2)
C 0 A 010 Q 11 — Shift

Add and Shift (Multiply with 1 1 0 1)
+ 1011
C 0 A 1101 Q 11 — Add (3)
C 0 A 110 Q 111 — Shift

Add and Shift (Multiply with 1 1 0 1)
+ 1011
C 1 A 0001 Q 111 — Add (4)
C 0 A 1000 Q 1111 — Shift

47

Unsigned Multiplication

```
  1011
 x1101
 0000
+1011   (1)
 1011
+0000   (2)
 01011
+1011   (3)
 110111
+1011   (4)
10001111
```

Initialize
C 0 A 0000 Q 110(1)

Add and Shift (Multiply with 1 1 0 1)
+ 1011
C 0 A 1011 Q 1101 — Add (1)
C 0 A 101 Q 1 11(0) — Shift

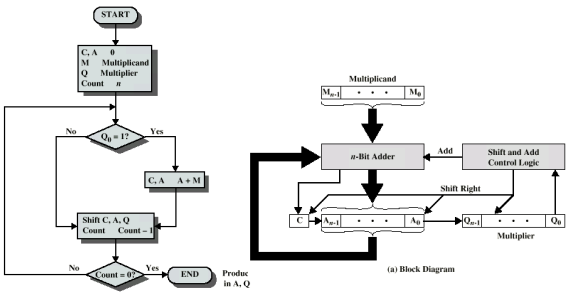
Add and Shift (Multiply with 1 1 0 1)
+ 0000
C 0 A 0101 Q 1110 — Add (2)
C 0 A 010 Q 111(1) — Shift

Add and Shift (Multiply with 1 1 0 1)
+ 1011
C 0 A 1101 Q 1111 — Add (3)
C 0 A 110 Q 111(1) — Shift

Add and Shift (Multiply with 1 1 0 1)
+ 1011
C 1 A 0001 Q 1111 — Add (4)
C 0 A 1000 Q 1111 — Shift

48

Unsigned Multiplication



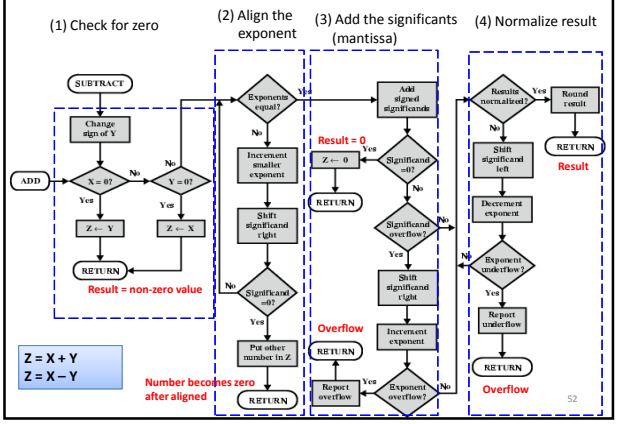
ALU: Multiplying Negative Numbers

- This does not work!
- Solution 1
  - Convert to positive if required
  - Multiply as above
  - If signs were different, negate answer
- Solution 2
  - Booth's algorithm (not covered)

Floating Point Addition & Subtraction

- Check for zeros
- Align significands (adjusting exponents)
- Add or subtract significands
- Normalize result

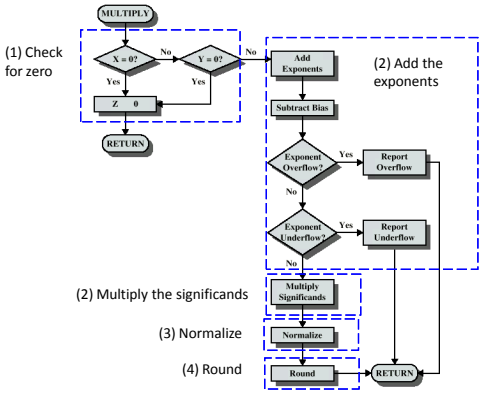
Floating Point Addition & Subtraction



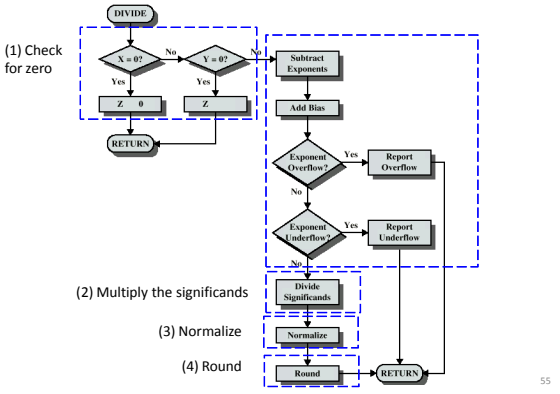
Floating Point Multiplication / Division

- Check for zero
- Add/subtract exponents
- Multiply/divide significands (watch sign)
- Normalize
- Round
- All intermediate results should be in double length storage

Floating Point Multiplication



Floating Point Division



55