

Computer Organisation and Architecture

01 October 2019 09:06

- Intro
- Data Representation
- Digital Logic
- Machine Code and Assembler
- Memory Systems
- I/O Mechanisms
- Processor Architecture

Coursework 40% due Wednesday week 9 term 1 / Thursday week 3 term 2
Exam 60%

Reading

Principles of Computer Hardware – A. Clements

Structured Computer Organization – A.S. Tanenbaum

Memory Mapped I/O

1. How they work
2. Advantages
3. Disadvantages

Three-state logic

There exists a third state – unconnected - when the enable line is low then the switch is connected.

The same address bus is used to address both memory and I/O devices

Memory (including register) on I/O devices are mapped to address values

A/O device can then operate as designed on the data given

When the CPU access a memory address that memory address may be in a physical memory typically RAM or be associated with the memory of an I/O device

Advantages

This process is simple and requires no extra hardware

Port I/O has a dedicated set of instructions dedicated to I/O and so is more complicated – too many instructions for different devices

CPU requires less internal logic which can help the design of the CPU be cheaper

Disadvantages

Portions of memory address space must be reserved

This is irrelevant with 64 bit processors but 16 bit and 32 bit processors are used – e.g. legacy and embedded systems

Polling

Most I/O are slower than the CPU

READ – is there a data to be read from the device?

WRITE - is the device ready to accept data?

Rather than having a microprocessor constantly check for input, we use 'busy-wait' polling which does things while we wait giving the appearance of multitasking

Advantages

Simple software – simple loop function is all that is needed

Simple hardware – support for notion of ready is all that is required

Disadvantages

Busy-wait polling wastes CPU time and consumes power – if you have a power constrained device

Polling when interleaved with other tasks can lead to significantly delays response to device

Handshaking

Unsynchornised – print this

Open-Ended – send signal when data sent is ready to print

Closed-Loop – data provided, data valid signal sent, printer ready signal sent back, print

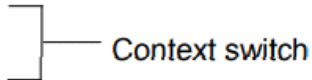
Interrupts – EXAM Q

IRQ Interrupt Request can be ignored or masked out

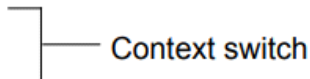
NMI – non-maskable inputs – cannot be masked out

1. External device signals interrupt

INTERRUPT RESPONSE

1. CPU completes current instruction
 2. Push PC onto Stack
 3. Push Status Register(s) onto Stack
 4. Load PC with address of Interrupt Handler
- 
- Context switch

RETURN FROM INTERRUPT

1. Pop PC from Stack
 2. Pop Status Register(s) from Stack
 3. Load PC with popped return address
- 
- Context switch

Asynchronous nature leads to fast response

No CPU time wasted / battery power

Disadvantages

All data transfer controlled by CPU

More complex hardware and software compared to polling

Direct Memory Access DMA

Problems with I/O

Control of the CPU is surrendered to the DMAC which can control the address bus, data bus etc.

This skips over the CPU bottleneck (CPU can still override)

Modes of operation

Cycle Stealing

Uses system buses when they are not being used by the CPU

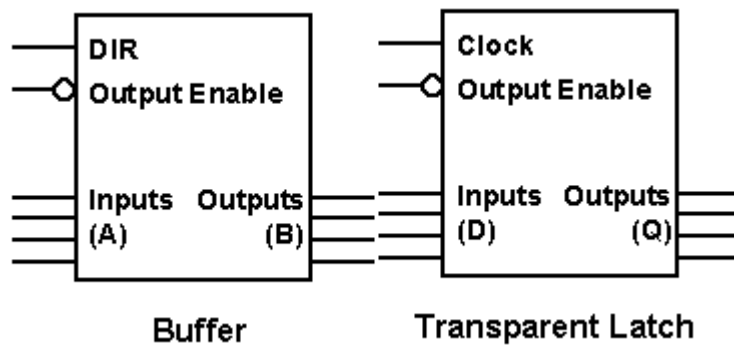
Burst Mode

Modes of Operation

DMA Organisation

Lab – 4 Bit Bus

23 October 2019 11:23



The 4 bit bus is created by linking a buffer to a latch

When output enable is 0, the output of the components is active

When it is 1, the output is blocked.

Direction 1 = A to B

Direction 0 = B to A

When the clock is high and the OE bar (of the latch) is 0, the latch is transparent

When the clock changes from high to low, the latch STORES THE INPUT

Then the latch is called stable. Q stores the state regardless of changes of state to D

4 lines buffer to latch inputs

4 switches to buffer

4 LED's to latch outputs

How to Store Values

- 1) Set latch to transparent
- 2) Modify inputs to desired values
- 3) Change clock to low
- 4) inputs are stored

Switching between inputs

When two buffers are connected to the latch, it is possible to use a clock to switch between the inputs. E.g. we take the values of switches 1-4, then a clock pulse changes the values being accepted to switches 5-8, then back to 1-4 etc etc.

- 1) Enabled OE bar for both buffers
- 2) This blocks both of them.
- 3) Disables OE bar for the desired buffer
- 4) Desired buffer is now outputting data to latch

We took the two OE bars lines and connected them to the Push Button (analogous to a clock pulse) and then ran one of the inputs through a NOT gate. This means that it is not possible to have both OE's at 0 (actively outputting) at the same time

Bases

01 October 2019 09:40

Why do we use hex or decimal?

- Working in decimal is tricky. One value is 3.3 bits.
- In hex, one value is 4 bits
- In octal, one value is 3 bits.
- This is easier to process
- $1111 \text{ (base 2)} = 15 \text{ (base 10)} = F \text{ (base 16)}$

Electrical Circuit Representation

01 October 2019 09:47

Binary gives the greatest possible noise immunity. Introducing more separations increases likelihood that interference can distort a value across a boundary, causing its value to be lost.

Transistor-Transistor Logic

0-0.8v = 0

2.4-5v = 1

2.3 would be 1 – unspecified inputs float high

Word – the number of bits a machine can process simultaneously.

MSB – most significant bit

LSB – least significant bit

Intel 4004 (1971) had a 4 bit processor

G5 e.g. Intel Core (2004) has a 64 bit processor

Why do we use hex or decimal

Working in decimal is tricky. One value is 3.3 bits.

In hex, one value is 4 bits

In octal, one value is 3 bits.

This is easier to process

1111 (base 2) = 15 (base 10) = F (base 16)

To convert to a different base, divide the denary by the new base, and record the remainder

Converting binary to hex

Convert each 4 bit block to hex

e.g. 1001 1010 = BA

Logic Gates, Circuits and Truth Tables

08 October 2019 09:14

https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs132/03_digital_logic_-_4up.pdf

All logical functions can be created using AND OR and NOT

All logic gates can be created using NAND and NAND inverters (single input nands)

It follows that all logical functions can be created using NANDS

Logical expressions can be simplified to reduce complexity, reducing the number of gates used for the implementation

There are two ways to simplify logical expressions

Boolean Algebra (manipulation of algebraic representation)

Karnaugh Maps (simplifying the sum of products form graphically)

Exclusive or cannot have three inputs

Any two bit system has 16 possible combinations using the available gates

Sum of products form

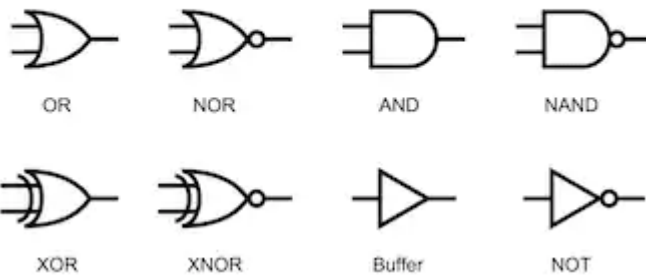
Obtained from the truth table

Just map the input values directly to the output

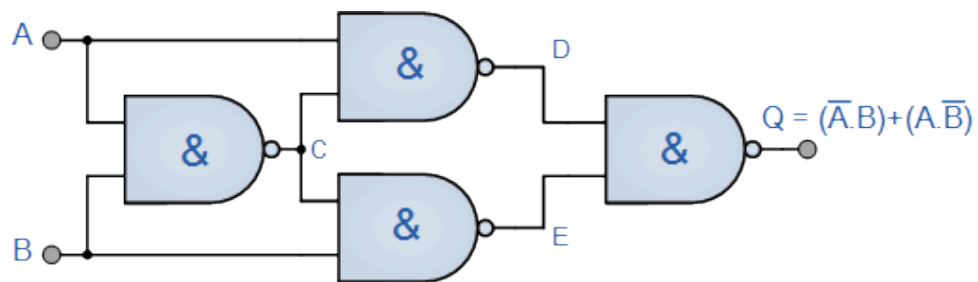
e.g. $a, b, c = 0, f = 1 = (\text{NOT } A) \cdot (\text{NOT } B) \cdot (\text{NOT } C)$

So function is true when that condition is met

Logic Gate Symbols



Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\bar{A} = 0$	$A + \bar{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\bar{B}$



NAND Gate Realisation

Karnaugh Maps

09 October 2019 12:28

https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs132/03_digital_logic_-_4up.pdf

2D truth tables

Used to see the simplest boolean expressions of a few variables from the location of 1s on the maps.

A three variable function can be represented by a 4x2 rectangle or a 2x4 rectangle

Take the values from the truth table and populate the karnaugh map

Wrap-Around

Left most column is adjacent to right most column, and top row is adjacent to bottom row

Groupings may overlap

Minimum logical expression is obtained on minimum number of groups

Number of elements in a group must be a power of two

e.g. 1,2,4,8

C	AB			
	00	01	11	10
0	1	1	0	1
1	1	0	0	1

$$\Rightarrow f = \bar{B} + \bar{A}C$$

C	AB			
	00	01	11	10
0	1	1	0	1
1	1	0	0	1

$$\Rightarrow f = \bar{A}\bar{B} + A\bar{B} + \bar{A}C$$

Correct, but not minimal. The top result was simpler - don't forget to wrap around!

To extract information, check what is true for the variables in the group e.g. if A is always the same, then it will be part of the definition. OR this with the other definitions

The larger the individual grouping, the smaller the term that you will get out of it

If the table is complex and there are more 1's than 0's, you can find the value for the 0's to find NOT F then invert the answer

CD \ AB	AB			
	00	01	11	10
00	0	1	1	1
01	X	1	1	1
11	1	1	1	1
10	0	1	1	1

$$f = A + B + C.D$$

CD \ AB	AB			
	00	01	11	10
00	0	1	1	1
01	X	1	1	1
11	1	1	1	1
10	0	1	1	1

$$f = A + B + D$$

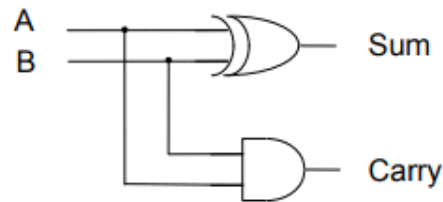
Occasionally a combination of inputs can't happen, or if it can it doesn't affect the output. These are showed by X. Use these to your advantage to find bigger groupings as shown above, in order to create simpler expressions.

Combinatorial Logic

09 October 2019 12:44

Half Adder

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



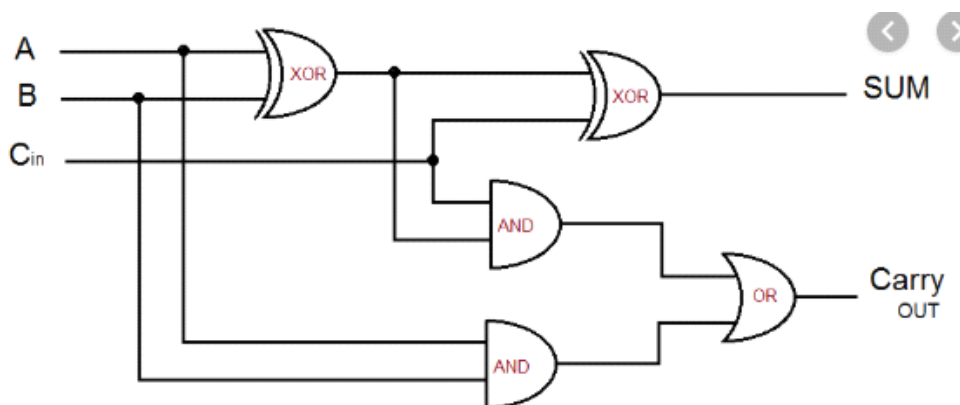
Full Adder

The full adder circuit is a combination of two half adders combined with an OR gate.

It takes 3 inputs – 2 bits to sum and a carry, and adds them. The ability to take a carry and output a carry means that the full adder circuits can be chained to create an n-bit full adder. These circuits are one of the fundamental building blocks of the ALU

The circuit contains two XOR gates, two AND gates, and an OR gate.

- The XOR from the first adder goes to the second XOR and the second AND.
- The AND from the first adder just goes to the OR (to create the carry)
- The carry in goes to the second XOR and the second AND

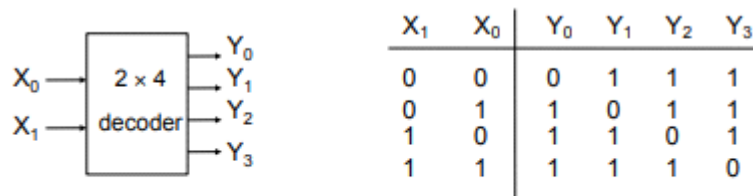


Decoders

A decoder has binary input pins and one output pin per possible input state.

A decoder will activate one output pin corresponding to the current input state

E.g. 2 input pins would translate to 4 output pins (00,01,10,11)



Note that the two inputs select which output is active

The system is active low, i.e., active = 0, in this case

NOTE: Some systems are active low (0 = inactive, 1 = active) and some are active high (0 = inactive, 1 = active)

Decoders are often used to address unique memory locations in a microprocessor system

Encoders

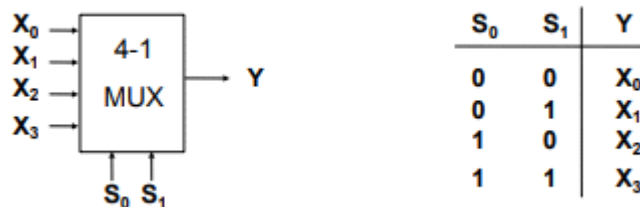
The opposite of an encoder. Encodes a set of inputs into a defined representation

Multiple inputs pins, of which only one should be active at a time

Small number of output pins

Multiplexers (MUX)

Output is a selected input



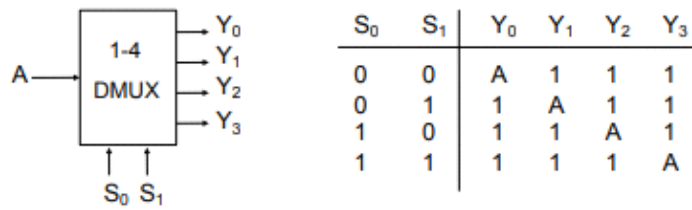
The S inputs determine which X is output to Y. e.g. when the S1 and S2 are both 1, (3 in binary) X3 is active

Common Applications

- Source selection control e.g. audio systems with multiple inputs (aux, bluetooth)
- Sharing communication lines between multiple senders
- Parallel to Serial conversion

De-Multiplexers (DE-MUX)

Allow an input to appear on any one of the outputs (reverse of a multiplexer)



A 1 represents inactive: therefore active low

Common Applications

Share one communication line between multiple people (requires both MUX and DE-MUX)

Sequential Logic Circuits

15 October 2019 09:25

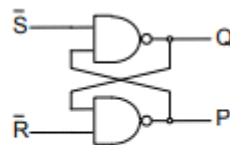
1. A sequential logic circuit is a circuit whose outputs are logical function of not only its inputs, but also its current state
2. A state dependent combinatorial logic circuit
3. A combinatorial logic circuit with a memory element

Flip Flop

A circuit whose output loops back to its input

\bar{S}	\bar{R}	Q	P
0	0	X	X
0	1	1	0
1	0	0	1
1	1	no change	

← hazard condition



The flip-flop is able to **STORE** a value when both $\bar{S} = \bar{R} = 1$

Q is Set (to one) when NOT S = 0 (and NOT R = 1)

Q is Reset (to zero) when NOT R = 0 (and NOT S = 1)

If NOT S = NOT R = 1 then Q does not change

D-Type Latch and D-Type Flip-Flop

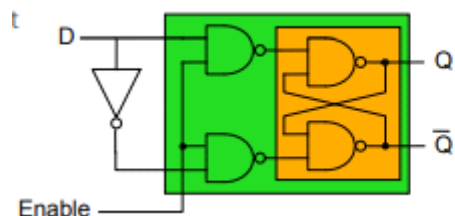
Uses a modified form of the flip-flop circuit

An edge triggered 1 bit memory circuit

Output can only change when the enable line is high

There are other types of flip-flops:

1. Delay Flip Flop (D-Type)
2. Toggle Flip-Flop (T-Type)
3. JK type



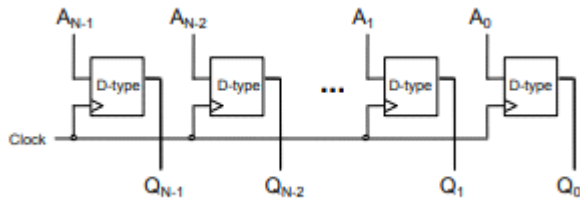
Enable	D	Q	\bar{Q}
0	0	Q	\bar{Q}
0	1	Q	\bar{Q}
1	0	0	1
1	1	1	0

Essentially a 1 bit memory circuit

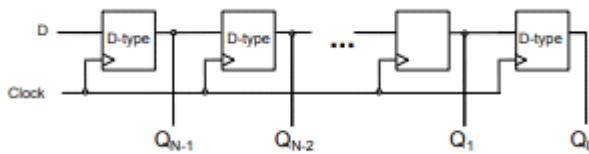
Registers and Shift Registers

23 October 2019 11:48

A register is a circuit that stores bits using D-type Flip Flops. Each flip flop holds one bit and outputs to Q regardless of change of input D, until another clock pulse at which point the flip flops will take on the current value of D and output it to Q

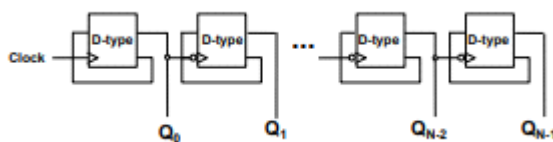


Shift Registers



When a clock pulse occurs, the values of the bits shift to the right.

N-Bit Counter



The circles on the inputs to the D-types (except left most) indicate inverters
Each flip flop is clocked only on a transition from high to low (falling clock edge) (except left most)

N Bit Subtractor

16 October 2019 11:20

To convert an N bit full adder to a subtractor, we simply add a control input Z, such that:

$$Z = 0 \Rightarrow A = A + B$$

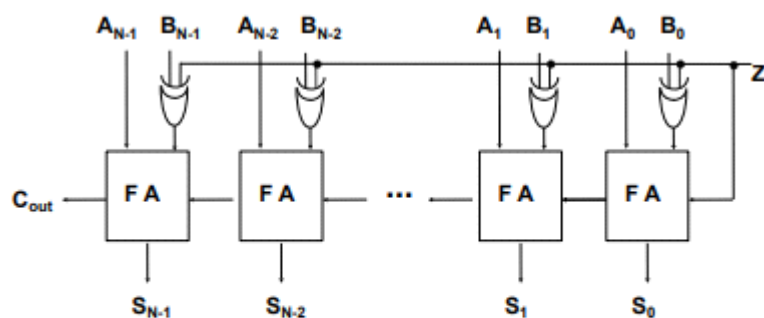
$$Z = 1 \Rightarrow A = A - B$$

Remember that $A + B = A + (-B)$

We can invert the N-bit binary number by flipping the bits and adding one

This is done by doing $Z \text{ XOR } B$

Add 1 (carry in)



$$Z = 0 \Rightarrow \text{Adder}$$

$$Z = 1 \Rightarrow \text{Subtractor}$$

A Carry out would indicate an overflow error.

Assembly Intro

16 October 2019 12:15

1. Microprocessor Fundamentals
2. Register Transfer Language
3. Assembly Language
4. Subroutines and Stacks
5. Addressing Modes

CPU

A CPU controls and performs instructions.

CPU's are commonly housed on a silicon chip known as a microprocessor

- ALU - performs arithmetic and logic
- Control Unit – Decodes program instructions and handles logistics for the execution of decoded instructions

CPU continuously performs a FETCH DECODE EXECUTE cycle

Instruction cycle takes place over several CPU clock cycles

Components of a FDE Cycle

ALU

CU

Program Counter PC – Stores memory address of the next instruction to be executed. Generally incremented after each instruction is executed

Instruction Register IR – contains the most recent instruction fetched

Memory Address Register MAR -

MDR – Contains data fetched from memory or data ready to be processed

Registers

22 October 2019 16:35

Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU. The registers used by the CPU are often termed as Processor registers.

A processor register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).

The computer needs processor registers for manipulating data and a register for holding a memory address. The register holding the memory location is used to calculate the address of the next instruction after the execution of the current instruction is completed.

The register specifics here relate to a 68k chip

Data Registers

Store frequently used values / intermediate results on chip.

Strictly speaking we would only need one register on chip, but the advantage of using many chips is that more data values can be stored, decreasing the amount of times data needs to be fetched from external memory

Registers can be treated as long, word or byte (32 / 16 or 8 bit)

Status Register

Consists of two 8 bit registers

Various status bits that are set or reset upon certain conditions arising in the ALU.

X = Extend

N = Negative

Z = Zero

O = Overflow

C = Carry

Address Registers

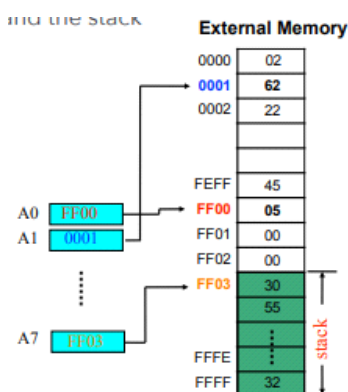
Used as pointer register in the calculation of operand addresses

Also can be used but the processor as a system stack pointer to hold subroutine return addresses

Operations on addresses do not alter status register condition code register (CCR)

ALU has the capacity to incur changes in status

This image shows the relationship between the address registers A0-A7 and external memory



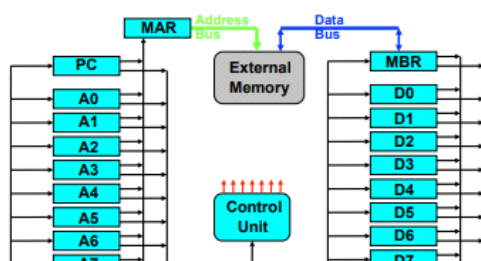
Stack Pointer

- A7 (the address register acting as stack pointer in this architecture) points towards an area of memory called the system stack. It points to the next free location.
- The stack is a LIFO structure
- The stack provides temporary storage of essential processor states, e.g. the return addresses and registers, during subroutine calls and interrupts.

Program Counter

A 32 bit register that keeps track of the address at which the next instruction will be found. In simple terms, it points to the next instruction in memory.

When the current instruction has been read, the PC is incremented to point to the next instruction



This is the internal architecture of the 68008 processor.

Why are there two ALUs?

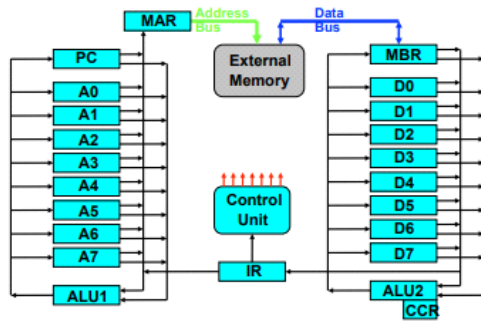
Why is the address bus one way?

What connections does the Control Unit have?

What is the purpose of connecting the Instruction Register directly to the Control Unit?

Why does only one ALU have a CCR?

What is the roll of the CCR?



This is the internal architecture of the 68008 processor.

Why are there two ALUs?

Why is the address bus one way?

What connections does the Control Unit have?

What is the purpose of connecting the Instruction Register directly to the Control Unit?

Why does only one ALU have a CCR?

What is the role of the CCR?

Register Transfer Language

16 October 2019 12:39

Used to describe the operations of a microprocessor as it is executing instructions

e.g. $[MAR \leftarrow PC]$ means transfer contents of program counter to MAR

Computers memory is called main store

The contents of memory location 123 is $[MS(123)]$

Do not confused RTL with ASSEMBLY instructions

Instruction Cycle

We can use RTL to represent stages of the FDE cycle

Fetch Stage

$MAR \leftarrow PC$

$PC \leftarrow PC + 1$

$MBR \leftarrow [MS(MAR)]$

$IR \leftarrow MBR$

$CU \leftarrow [IR(OPCODE)]$

This is the RTL for a Macro Fetch

Architectures

22 October 2019 09:19

Harvard Architecture – 2 separate memories for instruction and data

Von Neumann – 1 single memory for both instructions and data

Harvard can be argued to be faster as it has separate buses and so operations can be parallelised

Addressing Modes

22 October 2019 09:22

In 68008 systems data can be located in a data register, within the instruction itself or in external memory

- Provide data directly
- Specify exactly where data is
- Specify how to go about finding data

68008 Addressing modes:

- Data or address register direct
- Immediate addressing
- Absolute addressing

"Herse £100" - literal value

"Get the cash from room 19" - direct address

"Go to room 23 and they'll tell you where to get the cash" - indirect addressing

"Go to room 42 and get the cash from the 5th room on the right" - relative addressing

Direct

Move D3, D2

Moves the value in D3 to the D2 register. Also works for sending data to address registers

Immediate

Move.b #\$42, D5

In RTL this would be [D5] <- \$42

Absolute

Operand specifies the location in memory explicitly, meaning that no further processing is required

Move.l D2, \$7FFF0

[MS(7FFF0)] <- [D2]

Indirect

Address register indirect

Move (A0), D3

Address register indirect with offset

Move 7F(A1), D3

Post-Incrementing Address Register Indirect

Move.b (A0)+, D3

Pre-Decrementing Address Register Indirect

Move.b -(A0), D3

Indexed Addressing

Move.l 1F(A0, A1), D3

C Programming

23 October 2019 12:17

Data Types

C is a system programming language.
Used for operating systems, drivers, embedded systems.

It spawned a family of C type languages like java

There are no objects in C

Return 0 from functions to indicate successful execution

There are only 4 basic data types in C
Char, int, float, double
8, 32, 32, 64 bit respectively
Ints can be controlled with two qualifiers, long and short.

There are also two more qualifiers for integer types, unsigned and signed. (signed allows for negative numbers but reduces the range of values available)

No explicit Boolean in C
Booleans are implemented and evaluated as integer values

0 = false and non-zero = true

Arrays

Allocated contiguous blocks of memory
e.g. `int a[10]` allocates space for integers in one contiguous block

Arrays are not objects and so you can't access properties like length.
The length of the array must be explicitly recorded and tracked

The name of the array variable can be used as a pointer

Iteration
Easiest way to iterate over a C array is with an index variable
e.g. a for loop for (`i = 0; i < arrayLength; i++`)

Declare variables before the for loop and not in the loop

Strings

Strings are implemented using an array of characters but with an additional terminating character at the end (`\0`)

e.g. `"hello" = {'h','e','l','l','o','\0'}`

There is a string library that provides a range of string functions


```
#include <string.h>
```

All string functions in the C libraries will use the '\0' character to detect the end of a string

Control Flow – Conditionals

If, while, do are similar to those found in java

Conditions must produce an integer boolean results

If your condition for an if statement was $2 \% 2$ then it would not run because this equals 0

Pointers

References to locations in memory

One of the most common sources of bugs in software

Remember

- *ptr – look in the address pointed to by ptr
- &v – get the address at which v is stored

Compilers can't detect all bugs – inspect code thoroughly

Common mistakes

Un-Initialised Pointers

Not initialising a pointer will create a seg fault because the uninitialized pointer has a value of 0 by default

Out of bounds memory access

A pointer is used to access an array and pushed outside of the array's bounds
This either corrupts memory or gives a seg fault

We can use pointer arithmetic to move a pointer – this can be dangerous

Memory Management

C requires that all memory is managed.

Stack Memory

Local variables declared within a function are placed on the stack, the space they use is automatically free when the function returns

Memory on the heap

Non-local variables or arrays with an unknown size at compile time should be placed on the heap

Memory must be allocated and freed manually

Memory is allocated through use of the malloc function

```
Void* ptr = malloc(bytes)
```

Memory is freed by the user of free
`Free(ptr);`

Trying to free a null pointer will cause a segfault

Trying to free a pointer that wasn't returned by malloc will cause a segfault too

Why use the heap?

If the desired array is too big for the stack

Array size is not known until run time

More general purpose

Compilation

Compile c to object code: `Gcc -c myfile.c`

To link object code in an executable: `Gcc -o myprogram myfile.o`

To run the executable: `./myprogram`

Test – Digital Logic

27 October 2019 18:15

1. Subtract 9 from 13 in 8-bit wide two's complement. ----- [3]

	-128	64	32	16	8	4	2	1	
13	0	0	0	0	1	1	0	1	
-9 (flip bits and add 1)	1	1	1	1	0	1	1	1	
13 + (-9)	0	0	0	0	0	1	0	0	= 4
Carry bits	1	1	1	1	1	1	1		

2. Explain, with the aid of a diagram, the difference between combinatorial and sequential logic circuits. ----- [3]

A combinatorial logic circuit is a circuit whose output is a logical function of its inputs

A sequential logic circuit is a circuit whose output is a logical function of its inputs **and its current state** – a combinatorial logic circuit with a **memory element**



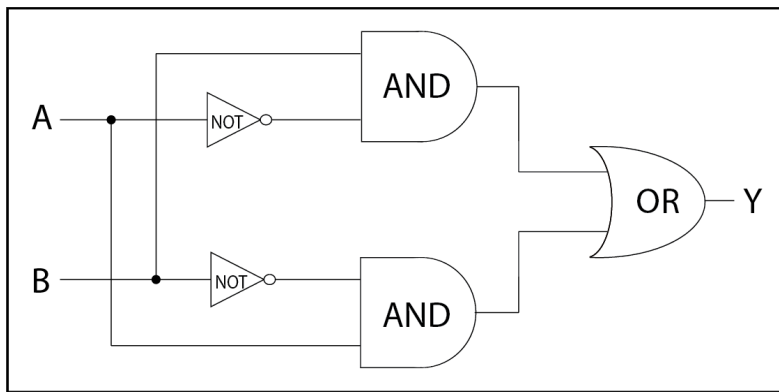
3. Show the truth table for an OR gate. ----- [3]

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

4. Show the truth table for an EX-OR gate ----- [3]

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	0

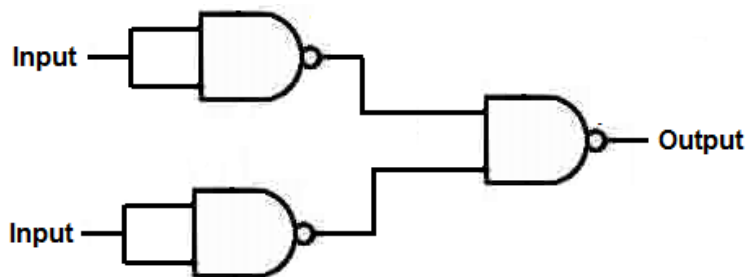
5. Design a circuit that implements the function of an EX-OR gate using only NOT, AND and OR gates. ----- [6]



6. Show the truth tables for a AND gate. ----- [2]

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

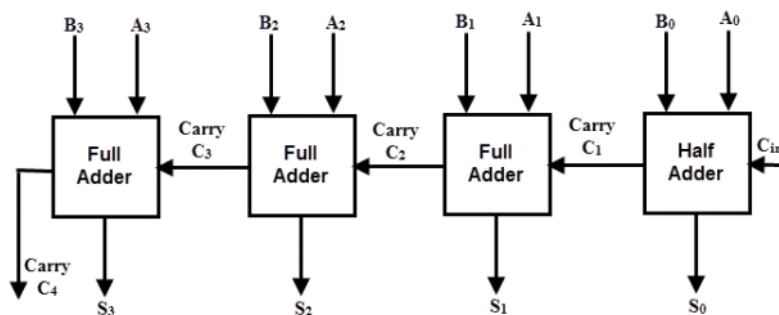
7. Design a circuit that implements the function of an OR gate using only NAND gates. ----- [3]



8. Show the truth table for a 1-bit full adder. ----- [5]

A	B	Cin	Sum	Cout
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

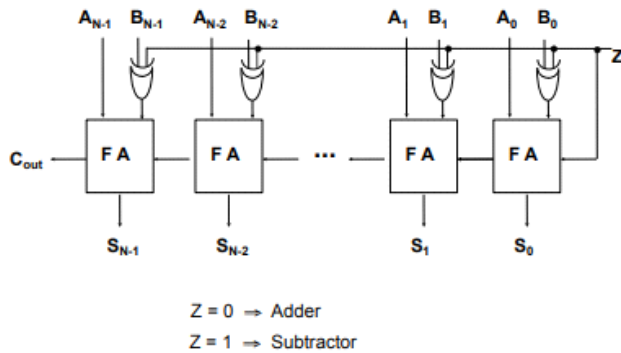
9. Design am N-bit Full Adder circuit. ----- [5]



10. Explain how an N-bit Full Adder circuit can be modified to form an N-bit subtractor circuit. ----- [4]

A control line (Z) needs to be added, which is input into an XOR gate along with the B input to each adder. The carry-in line needs to be connected to the control line so that it is set to 1 when Z is active (this effectively adds 1 to B). When the Z line is 1 and Carry in is set to 1, this has the effect of flipping all of the bits of B and adding one, which is how to turn a positive number into a negative number in two's complement. The adders can then simply perform an addition of the negative number to the positive number.

11. Design an N-bit Subtractor circuit. ----- [5]



12. Explain the function of a decoder, giving an example of where a decoder might be used. ----- [3]

A decoder takes a number of binary inputs and translates the number to the corresponding output pin. Activates one output pin corresponding to the current input state. Used for addressing unique memory locations in a microprocessor.

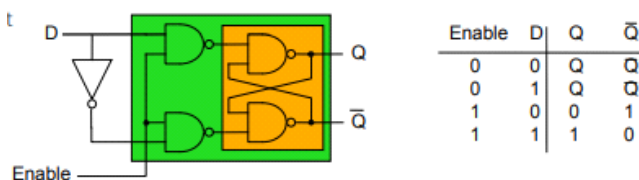
13. Explain the function of a multiplexer, giving an example of where a multiplexer might be used. ----- [3]

A multiplexer has a number of input pins X_1, X_2, X_3 etc, signal inputs (S_1, S_2 , etc) and an output pin (Y). The input that is connected to the output is determined by the signal of the S pins. E.g. if the S pins input the number 3, then the third input pin (X_3) would be output at Y .

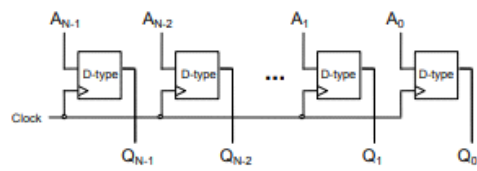
A multiplexer is used when there are multiple inputs to a system and only one can be active at a time, E.g. source selection for an audio system.

14. Explain, using an appropriate truth table or circuit diagram, the operation of a D-Type latch. ----- [4]

A D-type latch is a device that can be used to store one bit of information. The latch captures (or latches) the logic level which is present on the data line (D) when the clock input is high. If D changes while the clock is high, then the output (Q) follows D . When the clock input drops to 0, the output of Q is fixed and independent of the input D .



15. Show how D-type latches can be arranged to form an N-bit register, explaining the function of your circuit. ----- [5]



D-type latches can be lined up and connected to a clock. The output of the latches to Q remains fixed until a clock pulse is sent, at which point the latches simultaneously take on current logic level of their respective A lines.

Memory Hierarchy

05 November 2019 09:12

There is **no single memory system suitable for all of our needs** within the computer

The only condition is that a memory system must facilitate data modification and data detection

Many factors influence the choice of memory technology

- Frequency of access
- Access time
- Capacity required
- Cost per bit

The **designers dilemma** is to choose between low cost and high capacity or high performance

The memory hierarchy:

1. Registers – ultra fast as part of the CPU
2. Cache
3. Main store – fast program and data memory
4. Magnetic disk – non-volatile program and data storage
5. Optical Disk

Typical exam question: Describe and explain the motivation of the memory hierarchy

Discuss: **economics, designers dilemma, different requirements**

Small sequential loops, subroutines, operations on arrays, table and other structures mean that the locations in memory which will be accessed are somewhat predictable.

Temporal Locality

if a particular memory location is referenced, it is likely that the same location will be referenced again in the near future

Spatial Locality

if a particular memory location is referenced, it is likely that nearby memory locations will be accessed in the near future

For example, x in a for x loop can be stored near the processor because it is likely to be used again

It has therefore been shown that over 90% of data access occurs within two kilobytes of the PC

The two dominant things we think about when thinking about improving performance in computing

- **Parallelism**
- **Cache**

Cache

05 November 2019 09:22

- Uses the principles of temporal and spatial locality
- Stores the two kilobytes that have a 90% chance of being accessed in ultra-fast cache memory
- If data required by CPU is in the cache (a cache hit) then there is a big speed improvement

Problem

We want to use the fastest memory possible to store our data for best performance. Problems arise when we're using large datasets

Solution

Reuse data already in the cache as much as possible and prefetch data from main memory into cache before it is needed, masking load times

Moore's Law

The number of transistors will double roughly every 18 months

Cache Concepts

Caching read only data is straightforward; we don't need to consider the possibility that items will change, hence the copies across the hierarchy will remain consistent.

When caching writes, however, two strategies can be adopted

- **Write Through** – Updates the item in cache and writes through to update lower levels of the memory hierarchy
- **Write Back** – Only updates copy in cache, ensuring that blocks are copied back to memory when they are ready to be replaced

Cache Misses

$$h = \frac{\text{Number of times the words are in cache}}{\text{Total number of memory references}}$$

From this we can do $1-h$ to work out the miss rate

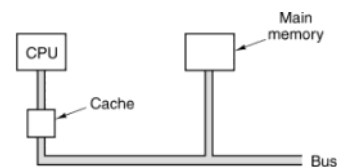
- Compulsory – a miss that occurs regardless of cache size. The first access to a block cannot be in cache, regardless of cache size, so the block must be retrieved
- Capacity – Misses occurring because the cache is not large enough to contain all blocks needed during the execution of a program
- Conflict – Misses occurring as a result of the placement strategy for blocks not being fully associative, which means that block may be discarded and retrieved
- Coherency – misses occurring due to cache flushes in multiprocessor systems

Cache Performance

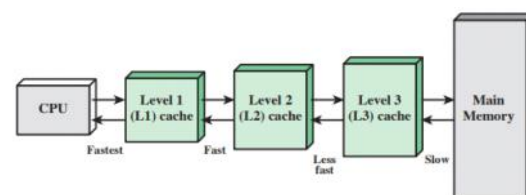
- Performance measures based solely on a hit miss rate don't take into account the cost of a cache miss which is the real performance issue
- Average memory access time is an alternative measure that accounts for the cost of a cache miss
- $\text{Average memory access time} = \text{Hit time} + (\text{Miss rate} \times \text{Miss Penalty})$
- Where hit time is the time to hit a in the cache and the miss penalty is the time taken to replace the block from memory
- Even average access time is still an indirect measure of performance as execution time an often be a better measure in real world situations

Multilevel Cache

Standard computers have 3 levels of cache and it has been shown that 3 is sufficient to maintain a near perfect hit rate assuming effective caching algorithms



execute typical instruction	1/1,000,000,000 sec = 1 nanosec
fetch from L1 cache memory	0.5 nanosec
branch misprediction	5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec



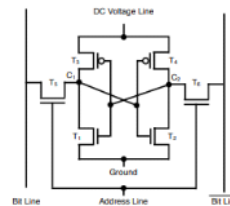
Memory Cell Organisation

07 November 2019 20:44

The most common form of main store is RAM – SRAM and DRAM

SRAM

Holds data as long as power is supplied – uses a configuration of flip flops and logic gates
Uses 4 cross-coupled transistors to establish a stable logic state

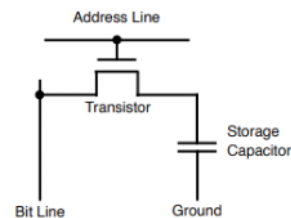


DRAM

Store data as charge on capacitors

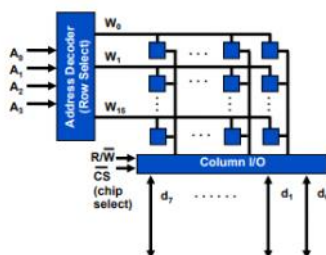
Capacitors naturally discharge, hence DRAM cells require periodic charge to maintain storage

The term dynamic refers to the natural leakage of charge that takes place even when power is supplied
Presence or absence of charge is interpreted as a logical 0 or 1



Write Operation	Read Operation
Voltage applied to the bit line (high for 1, low for 0)	Transistor turns on when address line is selected, allowing the stored charge to be fed out to the bit line and a sense amplifier
A signal is applied to the address line, allowing a charge to be transferred to the capacitor (data store)	Sense amplifier compares the capacitor voltage to a predetermined threshold value to determine if the cell contains a logical 0 or 1
	The read process discharges the capacitor, which means that it must be restored as well as refreshed, following the operation

Organisation of a Memory Chip



A 16-8 memory integrated circuit

16x8 denotes the words x bits – 16 words each of 8 bits

Choices for Memory Chip Organisation

128x8 would require:

- 8 data pins (word size)
- 7 address pins ($2^7 = 128$)
- Total of 15 IO pins + power etc

1024x1 would require:

- 1 data pin
- 10 address pins ($2^{10} = 1024$)
- Total of only 11 pins + power etc

But this results in a long, narrow cell array with a large address decode, which is an inefficient use of space

Noise

As feature sizes get smaller, the effect of noise increase

- Magnetic fields
- Thermal noise
- Transmission circuit noise#

Digital logic gives high noise immunity

Immunity collapses when the noise reaches a certain magnitude

Detecting Isolated Errors

- Usually caused by noise
- Could send the bit 3x – this is expensive

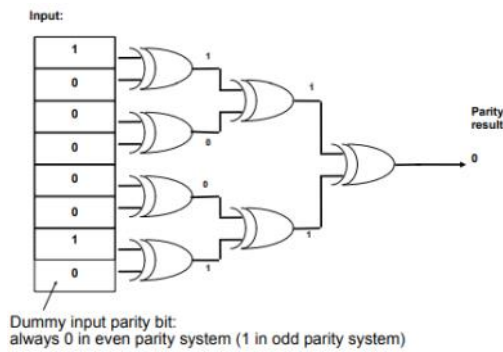
If the probability is low then the chance of having two errors is very low

Therefore we use **parity** bits

- Even parity system – the value of the extra bit is chosen to make the total number of logic 1s an even number
- Odd parity system – make the logic 1s odd

Parity bit value can be computer in software or hardware

Computing Parity in Hardware:



Burst Errors

In practice errors usually occur in bursts,

Parity can detect single, isolated errors, but an even number of errors close together leaves parity unchanged
Parity is therefore inadequate for burst errors.

Checksum

Checksums vastly improve our error detection capabilities and allow for the detection of burst errors

Error Correcting Codes

By extending to ECC and calculating both character and bit-column parity we can detect and fix some small errors

M	0	100	1101	
e	0	110	0101	
s	1	111	0011	
s	1	111	0011	
a	1	110	0001	
g	1	110	0111	
e	0	110	0101	
K	0	100	1011	

1. ← character parity

2. bit-column parity

At transmitter, we compute and transmit

- Computer parity for character and bit-column parity and transmit message with these added bits

At receiver, re-computer, compare and correct

- Computer character parity bits
- For each char, compare re-computed to received parity
- If not equal, error in row
- Re-compute bit-column parity
- Compare re-computed to bit column parity
- If not equal, error in column
- If single error detected, invert bit at (row, column)

✓	0	0	100	1101	M
✓	0	0	110	0101	e
✓	1	1	111	0011	s
x	0	1	110	0011	c
✓	1	1	110	0001	a
✓	1	1	110	0111	g
✓	0	0	110	0101	e
	0	0	100	1011	
	0	0	101	1011	

✓✓x ✓✓✓✓

Key:
1101 – received message bits
0 – bit changed by noise
1011 – received parity bits
1011 – re-computed parity bits
✓✓x – comparisons

This ECC can detect and correct single errors

It can also detect, but not correct, certain combinations of multiple error

RAM

05 November 2019

09:43

SRAM – Static Ram	DRAM – Dynamic Ram
SRAM uses a flip flop as storage element for each bit	DRAM for each bit, use the presence or absence of charge in a capacitor to denote a 1 or 0
SRAM typically provide better read and write times than DRAM - Operates at the speed of the flip flop technology within which is very fast	Capacitor charge leaks away over time. Requires periodic refreshing but is cheaper than SRAM
Both are volatile – power must be continuously applied	Both are volatile – power must be continuously applied
	Dynamic memory cells are: <ul style="list-style-type: none">• Generally simpler and more compact• Allows great memory density• Cheap to produce than SRAM

RISC vs CISC

13 November 2019 13:19

Reduced Instruction Set Computers are comprised of a small number of general instructions

$\text{MIPS rate} = f * \text{IPC}$

Where f is the processor clock frequency and the IPC is the average number of instructions executed per cycle

However by increasing f we increase power usage and heat dissipation problems

Alternatively we can use multithreading