

# Computer Architecture and system programming

Kristoffer Klokke

2022

# Contents

<b>1</b>	<b>The basics</b>	<b>3</b>
1.1	Structure and Function . . . . .	3
1.2	Gates, memory cells, Chips, and Multichip modules . . . . .	4
1.3	Processor architecture . . . . .	5
1.4	Embedded systems . . . . .	5
<b>2</b>	<b>Performance</b>	<b>5</b>
2.1	Measuring performance . . . . .	6
<b>3</b>	<b>Digital logic</b>	<b>8</b>
3.1	Boolean algebra . . . . .	8
3.2	Karnaugh maps . . . . .	9
3.3	Quine-McCluskey method . . . . .	10
3.4	Circuits . . . . .	11
3.4.1	Multiplex . . . . .	11
3.4.2	Decoders and encoders . . . . .	12
3.4.3	Read-only Memory . . . . .	12
3.4.4	Sequential circuits . . . . .	12
3.4.5	Programmable logic devices . . . . .	12
<b>4</b>	<b>Instruction sets</b>	<b>13</b>
4.1	Machine instructions . . . . .	13
4.2	Types of operands . . . . .	16
4.3	Types of operations . . . . .	16

# 1 The basics

Computer architecture: attributes of a system visible to the programmer, such as instruction set architecture (ISA) which defines opcodes, registers, instruction and data memory.

Computer organization: operational units and their interconnections, which are the behind the scenes of the architecture.

## 1.1 Structure and Function

A computer can perform 4 basic functions:

- Data processing - manipulate data in some form
- Data storage - in every computer some form of storage is needed even if it just temporary
- Data movement - data movement can be in many forms but most clear is the data movement from the input/output (I/O) referred to as peripheral
- Control - a control unit which can orchestrate the performance and functional parts of the computer

This therefore creates a computer structure of: CPU, Main memory, I/O, System bus.

The CPU are here a unit consisting of:

- Control unit - Control the operations sent to the CPU
- Arithmetic and logic unit (ALU) - perform the data processing
- Registers - storage for the CPU
- CPU interconnection - communication between the different units in the CPU

Some processors have multiple levels of cache where the higher level the faster yet smaller cache.

Some CPU may also have multiple cores which consist of:

- ISU (instruction sequence unit) - controls instructions sequence and allows for an out-of-order (OOO) sequence

- IFB (instruction fetch and branch) and ICM (instruction cache and merge) - These two subunits contain the 128-kB instruction cache, branch prediction logic, instruction fetching controls, and buffers.
- IDU (instruction decode unit) - fed from the IFU buffer it parses and decodes architecture operation codes
- LSU (load-store unit) - contains L1 data cache and controls data flow between L1 and L2 cache
- XU (translation unit) - Translate logical addresses into physical addresses
- PC (core pervasive unit) - Collects instrument data and errors
- FXU (fixed-point unit) - executes fixed point arithmetic operations
- VFU (vector and floating-point unit) - Handles all binary and hexadecimal floating point operations and fixed-point multiplication
- RU (recovery unit) - Keep a copy of the complete state in case of recovery
- COP (dedicated co-processor) - data compression and encryption functions for each core
- L2D - data cache for memory traffic
- L2I - instruction cache

## 1.2 Gates, memory cells, Chips, and Multichip modules

The only two required components for a digital computer are: gates and memory cells

A gate is a component which implements a boolean or logical function, ex AND gate.

A memory cell can be in two states at all time on or off and in this way save a bit.

A transistor is the electric based implementation of a gate or memory cell

### 1.3 Processor architecture

The Intel x86 by the complex instruction set computers (CISCs).

Unlike ARM which is based on reduced instruction set computer (RISC).

### 1.4 Embedded systems

These are system which are general purpose, but system where hardware and software (embedded system (OS)) are coupled together.

Theses system is found everywhere, and often working with the external environment via sensors.

Due to the software only having one purpose they are more efficient in both energy and processing power.

An embedded system may use a general purpose chip but most use a dedicated processor with specific number of needed tasks.

These dedicated chips often take form in microcontrollers which are sos called computers on a chips, small chips which have the same requirements for the 4 basic functions of a computer.

Deeply embedded systems are microcontrolelrs with burnt in programs and no interactions with the user.

## 2 Performance

In order to achieve the processing power of today different methods are being used to keep task comming to the CPU

- Pipelining - An execution of an instruction has multiple parts like: fetching instruction, decoding opcode, fetch operands and so on. Pipelining handles multiple executions by handling a different parts in every task.
- Branch prediction - By looking ahead in the instruction code, a prediction of needed instructions and buffers can be fetched beforehand.
- Superscalar execution - Multiple instructions are performed every processor clock cycle.
- Data flow analysis - By observing which instructions depend on other instruction result, a new order of instruction are made.
- Speculative execution - By using branch prediction and data flow analysis, the CPU speculates on upcoming instruction and execute them.

To create a new and faster CPU there are three approaches:

- Increase speed by reducing size of the chip, and therefore reducing the travel time of information
- Speed up cache size, to reduce to waiting time on slow data transformation
- Change processor organization and architecture to allow for better parallelism

But by increasing the speed and lowering the size of transistor, it creates new problem such as: Power density becoming higher and making it harder to dissipate heat, slower electron flow due to smaller connection which creates more resistance, Memory access speed which is a common constraint for CPUs.

Therefore a more modern solution is multi core processor designs, such more cores with shared cache can archive more speed.

Amdahls law describe how multicore can speedup a process as followed

$$Speedup = \frac{1}{(1 - f) + \frac{f}{N}}$$

Where  $f$  is the code which can infinitely be parallelizable and  $N$  is the number of cores.

But this should be taken with a grain of salt due to in a real environment other processes are able to make use of extra cores in case of a non parallelizable task.

A simple way to measure required speed is Little's Law which is

$$L = \lambda W$$

Where  $L$  is the average number of unit in the system at any time,  $\lambda$  is average rate of items which arrive per unit time and  $W$  is a number of unit time.

## 2.1 Measuring performance

Clock speed are a way of measuring the speed of electric pulses in the CPU measured in Hz, The time between a clock tick is called cycle time.

Average cycle per instruction (CPI) is the average number of cycles needed for every available instruction.

With this the process time can be calculated as

$$T = I_C \times [p + (m \times l)] \times \tau$$

$I_C$  is instruction count,  $\tau = 1/f$  where  $f$  is clock frequency,  $p$  number of processor cycles for decode and execute,  $m$  number of memory references,  $k$  ratio between memory cycle time and processor cycle time.

Often the millions of instruction per second (MIPS) is used which can be found with:

$$MIPS = \frac{f}{CPI \times 10^6}$$

Which also can be found in variations with floating point operations (MFLOPS) This is a flawed measurement due to different architectures like RISC and CISC where RISC will always have an advantage due to the reduced instruction set.

A good benchmark should be:

- Written in high level language to make portability high
- Is representative of a kind of programming domain
- Easily measured
- Wide distribution

SPEC is a standard for benchmarking which uses these terms:

- Benchmark - program written in high level and able to compile and execute on every computer which implements the compiler
- System under test - the tested computer system
- Reference machine - the reference scores from a chosen machine to compare current result to
- Base metric - strict guidelines for compilation in order to be able to compare results
- Peak metric - Optimized settings for the given system
- Speed metric - The total time of execute a compiled benchmark
- Rate metric - the number of tasks which can be completed in a given amount of time

## 3 Digital logic

### 3.1 Boolean algebra

Boolean algebra are algebra based on only the values 1 or 0.

It consist of variables and the operations AND ( $\cdot$ ), OR ( $+$ ) and NOT ( $\bar{b}$ ) in that precedence.

Another often usefull operators are XOR ( $\oplus$ ), NAND ( $\overline{b \cdot a}$ ) and NOR ( $\overline{a + b}$ ) Set operation may also be performed on sets of boolean, where union is or, intersect is and. When applies the operation is performed on each bit one by one.

And then the universal set will just be a set of 0's.

For more info, checkout my logical proposition repo.

These gates only have two output and one output except the NOT gate,

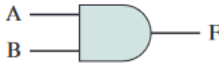
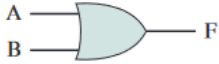
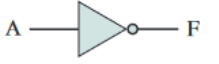



Name	Graphical Symbol	Algebraic Function	Truth Table															
AND		$F = A \cdot B$ or $F = AB$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \bar{A}$ or $F = A'$	<table><tr><th>A</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
NAND		$F = \overline{AB}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{A + B}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$F = A \oplus B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

Figure 1: Figure of gates and their logical operation

but any number of inputs is possible and some gates may have two outputs



where one is negated.

When designing a circuit the fewer amount of gates the simpler and to complete possible operation the following set combinations are possible:

- AND, OR, NOT
- AND, NOT
- OR, NOT
- NAND
- NOR

When writing circuit, it can be done in two forms, sum of products, where product expression are multiplied, and product of sums (POS) where sum are multiplied.

Both forms may not be the most simple form, but SOP uses only NAND, NOT and OR gates and POS uses only OR, AND, and NOT.

To simplify a circuit there are different methods

- Algebraic simplifications - This can be done with identities, which can simplify the expressions
- Karnaugh Maps - k-maps are a method which can help simplifying which variables are the out depended upon

### 3.2 Karnaugh maps

This method works by taking 2 to 4 variables from a truthstable or function. They are then setup in a grid such all possibilities are accounted for.

So for one side describing one variable there are 2 possibilities and a row which describes two variables there will be 4 possible outcomes.

For each row/column combination the function or truthstable are used to determine if the cell is 0 or 1.

Afterwards each 1 is circled in groups of powers of 2, so 1,2,4,8 or so on. A circle can not be cross or contain 0.

For each circle the depending non changing variables are used in an and form and may be negated if the input was a consisten 0.

For each circle the found AND expression is added with or to eachoter.

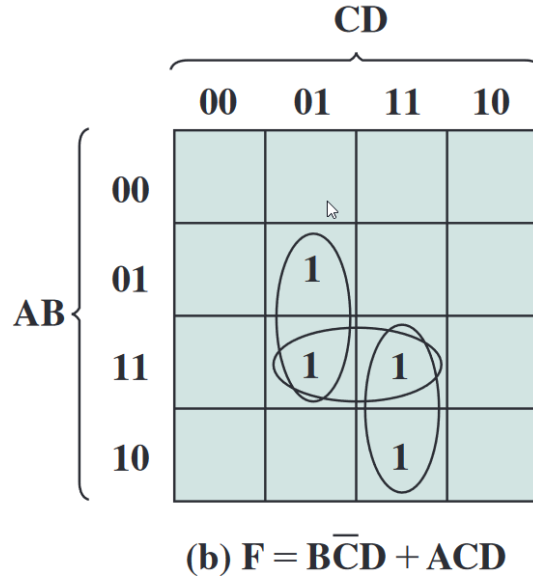


Figure 2: Example of kmap and the output function

### 3.3 Quine-McCluskey method

This is a more suitable method for SOP which have more than 4 variables. The method works by first creating a table with each collection of products on each row and in every column is the variables and in each cell are the needed value for the term to turn true.

The table is then ordered such the row with most 0's is at the top at the row with most 1's are at the bottom.

Then every row is compared to every row starting at the top, and if a row exist with only one column difference, the difference variable is eliminated and the rest of the variables are added to a new list.

Then every element in the list is done with same procedure, and new found objects are added to the list. Then the same step is repeated with every new element until there is no new elements.

Then every element from the list is added to a table as rows and the original terms are added as columns.

Then an X is placed in every cell where the row product is contained in the column. Then circle every X which is alone in a column, and square every X which are in a row with a circle.

Those rows with a marked X are now needed for the minimal expression.

Product Term	Index	A	B	C	D	
$\bar{A} B \bar{C} D$	1	0	0	0	1	✓
$\bar{A} B C \bar{D}$	5	0	1	0	1	✓
$\bar{A} B C D$	6	0	1	1	0	✓
$\bar{A} B \bar{C} \bar{D}$	12	1	1	0	0	✓
$\bar{A} B C D$	7	0	1	1	1	✓
$\bar{A} \bar{B} C \bar{D}$	11	1	0	1	1	✓
$\bar{A} B C D$	13	1	1	0	1	✓
$\bar{A} B C D$	15	1	1	1	1	✓

$\bar{A} \bar{C} D$   $\bar{A} B D$   $\bar{A} B C$   $\bar{A} B \bar{C}$   $B \bar{C} D$   $A \bar{C} D$   $A B \bar{C}$   $B \bar{C} \bar{D}$

Product Term	A	B	C	D	
$\bar{A} \bar{C} D$	0		0	1	
$\bar{A} B D$	0	1		1	✓
$\bar{A} B C$	0	1	1		
$\bar{A} B \bar{C}$	1	1	0		
$B \bar{C} D$		1	1	1	✓
$A \bar{C} D$	1		1	1	
$A B \bar{C}$	1	1		1	✓
$B \bar{C} \bar{D}$		1	1	1	✓

$B \bar{D}$   $B \bar{C} D$   $B \bar{C} \bar{D}$

	$A B C D$	$A B \bar{C} D$	$A B C \bar{D}$	$A \bar{B} C D$	$\bar{A} B C D$	$\bar{A} B \bar{C} D$	$\bar{A} B C \bar{D}$	$\bar{A} \bar{B} C D$
$B \bar{D}$	X	X			X		X	
$\bar{A} \bar{C} D$							X	⊗
$\bar{A} B C$					X	⊗		
$A B \bar{C}$		X	⊗					
$A \bar{C} D$	X			⊗				

Figure 3: Example of the method on

$$F = A B C D + A B \bar{C} D + A B C \bar{D} + \bar{A} B C D + \bar{A} B \bar{C} D + \bar{A} B C \bar{D} + \bar{A} \bar{B} C D + \bar{A} \bar{B} \bar{C} D$$

Which result in the list  $F = A B \bar{C} + A \bar{C} D + \bar{A} B C + \bar{A} \bar{C} D$

## 3.4 Circuits

### 3.4.1 Multiplex

Multiplex is a circuit of which a number of inputs label  $D_0, D_1, \dots, D_N$  is wired to an output  $F$ .

To controle which input determine the  $F$  value, the required number of selection inputs are used called  $S_1, S_2, \dots, S_N$ .

So for a 4 input it would require two  $S$  inputs.

### 3.4.2 Decoders and encoders

A decoder is a circuit with a number of output lines, with only one asserted at the time.

In general a decoder has  $n$  input and  $2^n$  outputs and can be useful for writing a specific sequence of bits according to a simple code.

An encoder will then be the inverse of the decoder.

### 3.4.3 Read-only Memory

As in the name this is memory, which can only be read from and are not programmable.

This is implemented using a decoder and a set of OR gates.

This is done by the a number inputs representing the placement of data and the OR gates each give out the value at the address.

### 3.4.4 Sequential circuits

Unlike combinational circuits as above a sequential circuits outputs are depending on current inputs and the current state.

The most simple form is a flip-flop, which is able to store one bit of data.

There are different types of flip flops with different properties The SR flip flop can not have both S and R be 1 but is the most simple, with a on (S) and off (R)

The D flip flop has a single switch for both on and off

The JK flip flop can have both inputs be 1 and will simply result in Q being 0

These flip flops can then be made in parallel forming a register, or by as a shift register which has flip flops in series and are sending data down the series at each clock cycle with only input at the front.

Another use case is a series of flip flops which creates a ripple counter or asynchronous counter, which when incremented the effect ripples through all the other flip flops.

Synchronous counters have the clock going into every flip flop. It can be observed that when counting in binary the first bit, simply flips on every count, the next bits flip when the right bit is 1.

### 3.4.5 Programmable logic devices

PLD are general-purpose chips.

There are different types of PLD

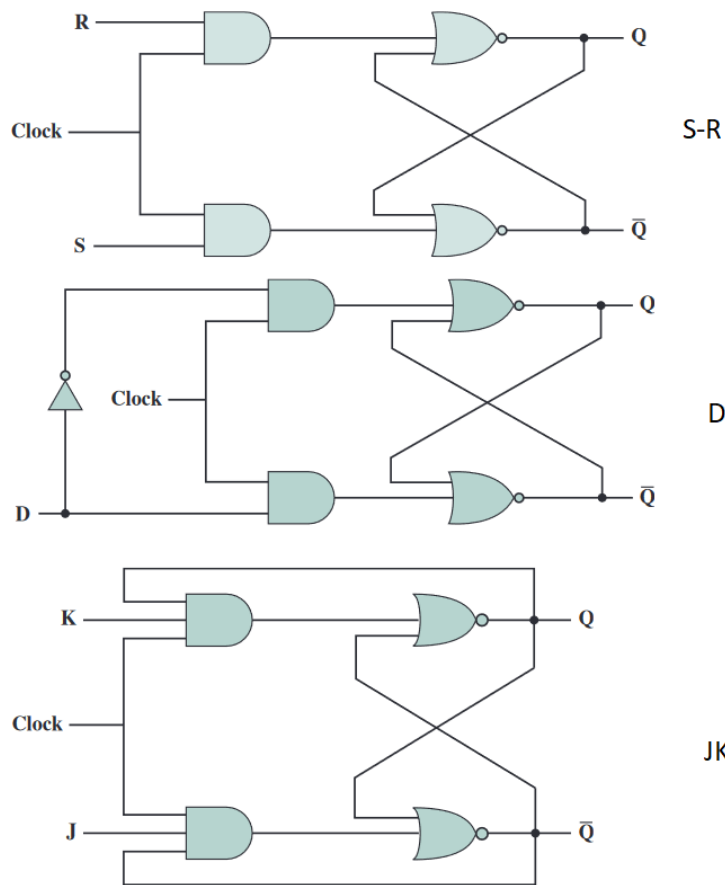


Figure 4: Different types of flip flops

- PLA - Programmable logic array, is circuit which allows a number of inputs in both normal and negated form, which goes into an array of and gates, which output are wired up to an OR array into the outputs. This takes advantage of SOP binary form
- FPGA - Field programmable gate array, is a circuit consisting of a logic block, which are programmable using flip flops, I/O blocks and interconnect which connects the I/O block to the internal logic blocks

## 4 Instruction sets

### 4.1 Machine instructions

A machine instruction consist of the following:

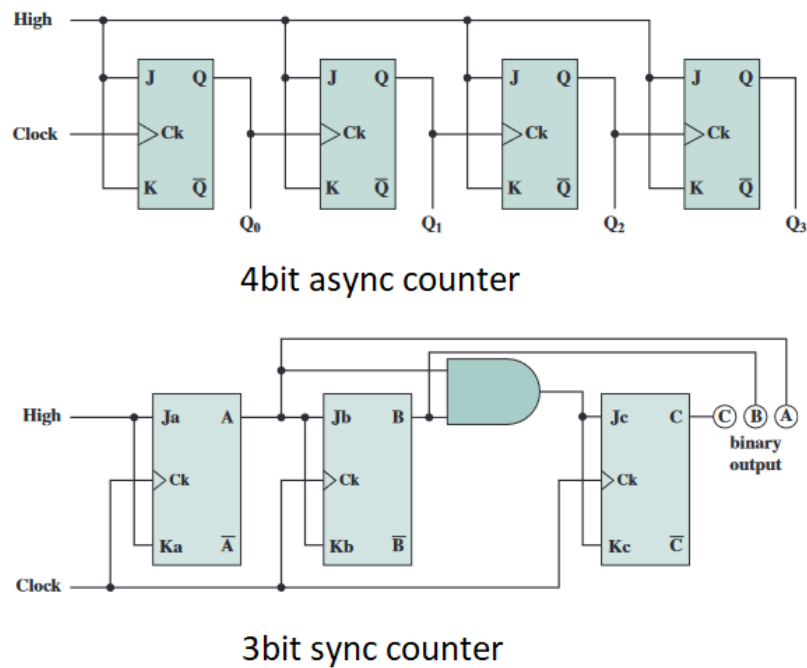


Figure 5: Two implementations of binary counters

- Operation code - the code which describe the operation to be performed called opcode
- Source operand reference - The operation may one or more source reference for the operation
- Result operand reference - The reference to the result of the operation if it exist
- Next instruction reference - The reference which hold the next instruction for after the execution

The next instruction reference and result reference can reference memory in different sources:

- Main or virtual memory
- Processor register - in some cases one or more registers may contain memory addresses which can be reference by the register name
- Immediate - The reference may be contained in the current instruction
- I/O device

When referencing instruction the opcode is most often represented as abbreviation called mnemonics, such as ADD

Instructions can be of the following types

- Data processing - Arithmetic and logic instructions
- Data storage - movement of data from and to registers and memory locations
- Data movement - I/O instructions
- Control - Test and branch instructions

Instructions may be designed to use

- 0 references - This will then use the stack for references
- 1 reference - Performs the instruction on and saves in a common accumulator register or something alike which the instruction is used upon
- 2 references - Performs the instruction and saves it in the first register
- 3 references - Performs instruction and first two references and saves in the last reference

When designing a set of instruction there are multiple questions have to be considered

- Operation repertoire - How many and which operations should be in the set
- Data types - Which data types should be available
- Instruction format - Instruction length number of addresses, size of various fields and so on
- Registers - Number of registers which should be referenceable and what their use should be
- Addressing - In which mode an address of an operand is specified

## 4.2 Types of operands

For numbers there are 3 different types

- Binary integer or binary fixed point - The classic integer in binary form
- Binary floating point - here the first bit mean the sign(1 = negative) the next 8 bits are the exponent and the next 23 are the mantisse for the 32 bit version
- Packed decimal - used to avoid a lot of conversion, such every decimal is represented with 4 bits, and (1101) means - and (1100) means +

For representing characters 8 binary bits can be used with standards by ASCII, which dictates what the different binary combination represent.

The x86 can deal with data types of 8 (byte), 16 (word), 32 (doubleword), 64 (quadword), and 128 (double quadword)

ARM processors support data types of 8 (byte), 16 (halfword), and 32 (word) bits in length.

## 4.3 Types of operations

A useful and typical categorization is the following:

- Data transfer - Calculate the memory address based on address mode, if virtual translate to real memory, determine if data is not cached and issue a command to the memory module.
- Arithmetic - Different kind of mathematic operations and may include data movement for the operation
- Logical - Logical operations such as XOR, right shift, left shift or rotate on binary data
- Conversion - Conversion between binary and decimal as well as operation conversion between 8 bit and such
- I/O - Instructions for data movement in and out of the system
- System control - Privilege function often reserved to operation system, such as alter register control or modifying storage protection key.



- Transfer of control - Operations for changing execution with: branching on condition (if and loops), skip on condition (skip out loop or flow), call a block of code and return to current code (function calling) is done by calling it and pushing parameters and return to stack in a stack frame.

When using conditions it refers to one of the architectures flags, which may be raised during instructions.

For x86 the call specifically does

- Push the return point on the stack
- Push the current frame pointer on the stack
- Copy the stack pointer as the new value of the frame pointer
- Adjust the stack pointer to allocate a frame

This can be done manually by instructions or by the ENTER instruction though it takes 10 cycles instead of 6.

MMX instructions are also exclusive to x86 and are instruction which can operate on multiple smaller data set by combining them into 32 or 64 bit chunks.

This allows the instruction to work in parallel on things like image processing where pixels are gathered in larger chunks.