Computer Architecture and system programming

Kristoffer Klokker

2022

Contents

1 The basics **5**
1.1 Structure and Function 5
1.2 Gates, memory cells, Chips, and Multichip modules 6
1.3 Processor architecture 7
1.4 Embedded systems 7

2 Performance **7**
2.1 Measuring performance 8

3 Digital logic **10**
3.1 Boolean algebra 10
3.2 Karnaugh maps 11
3.3 Quine-McCluskey method 12
3.4 Circuits 13
3.4.1 Multiplex 13
3.4.2 Decoders and encoders 14
3.4.3 Read-only Memory 14
3.4.4 Sequential circuits 14
3.4.5 Programmable logic devices 14

4 Instruction sets **15**
4.1 Machine instructions 15
4.2 Types of operands 18
4.3 Types of operations 18

5 Addressing modes and Formats **19**
5.1 Adressing modes 19
5.2 Addressing in x86 21
5.3 Addressing in ARM 22
5.4 Instruction formats 22
5.4.1 x86 instruction format 23
5.4.2 ARM instruction format 24

6 Assembly language concepts **25**
6.1 Assembly language elements 26
6.1.1 Pseudo-instructions 26
6.1.2 Macro definitions 27
6.2 Types of assemblers 29
6.3 Loading and linking 30

7 Assembly language x86 30
7.1 Directives . . . 31
7.2 Starting the code . . . 31
7.3 Registers . . . 31
7.4 System calls . . . 32
7.5 Jumps . . . 33
7.6 Stack and function calls . . . 33

8 Computer Arithmetic 34
8.1 Sign-magnitude representation . . . 34
8.2 Twos compliment . . . 34
8.2.1 Twos compliment arithmetic . . . 34
8.2.2 Twos compliment multiplication . . . 35
8.2.3 Twos compliment division . . . 36
8.3 Floating-point representation . . . 36

9 The Memory Hierarchy: Locality and Performance 39
9.1 Locality . . . 39
9.2 Memory systems . . . 40
9.3 Multilevel Memory Hierarchy . . . 40

10 Cache 41
10.1 Elements of cache design . . . 41
10.2 Types of cache misses . . . 42
10.3 Associative mapping . . . 42
10.4 Set Association mapping . . . 42
10.5 Replacement algorithms . . . 43
10.6 Write Policy . . . 43
10.7 Line size . . . 44
10.8 Inclusion policy . . . 44
10.9 Cache timing models . . . 44
10.10Performance modeling of multilevel memory hierarchy . . . 45

11 Internal Memory 45
11.1 Semiconductor main memory . . . 45
11.1.1 Error correction . . . 46
11.1.2 Synchronous DRAM . . . 46
11.1.3 DDR DRAM . . . 47

12 External Memory **47**
12.1 Magnetic Disk 47
12.1.1 Performance 48
12.2 RAID 49
12.3 Solid State Drives 49
12.4 Optical memory 50

13 C programming language **51**
13.1 IO 52
13.2 Memory allocation 52

14 I/O **53**
14.1 Programemd I/O 53
14.2 Interrupt-driven I/O 54
14.3 Direct Memory Access 55
14.4 Direct cache access 55
14.5 I/O channels and processors 55

1 The basics

Computer architecture: attrributes 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/ouput (I/O) refered to as peripheral
- Control - a control unit which can orchestrate the performance and functional parts of the computer

This therfore 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-streo unit) - contains L1 data cache and controls data flow between L1 and L2 cache
- XU (translatio unit) - Translate logical addresses into physical addresses
- PC (core pervasive unit) - Collects instrument data and errors
- FXU (fixed-point unit) - executes fixed point arithmetic operaitons
- 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 implments 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 implmentation 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 envirement 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 clocl 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 aproaches:

- 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 oragnization 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 becomming higher and making it harder to dessipate 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 hwo multicore can speedup a process as followed

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

Where f is the code which can infinitly be parallelizable and N is the number of cores.
But this should be taken with a grain of salt due to in a real envirement 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 Littles Law which is

$$L = \lambda W$$

Where L is the aver number of unit in the system at any time, λ 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 tic is called cycle time.
Average cycle per instruction (CPI) is the average weighted number of cycles needed for every avaliable instruction.

$$CPI = \frac{\sum_{i=0}^{n}(CPI_i \cdot I_i)}{I_c}$$

Where CPI_i is the number of operation for the instruction i and I_i is the number of the instruction. I_c is the total number of operations.
With this the process time can be calculated in two ways

$$T = I_c \cdot CPI \cdot \tau$$

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

I_C is instruction count, $\tau = 1/f$ where f is clocl 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:

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

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 hight level language to make portability high
- Is representive 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 implments the compiler
- System under test - the tested computer system
- Reference machine - the reference scores from a choosen machine to compare current result to
- Base metric - strict guidelines for compilation in order to be able to comapre results
- Peak metric - Optimized settings for the given system

- Speed metic - 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

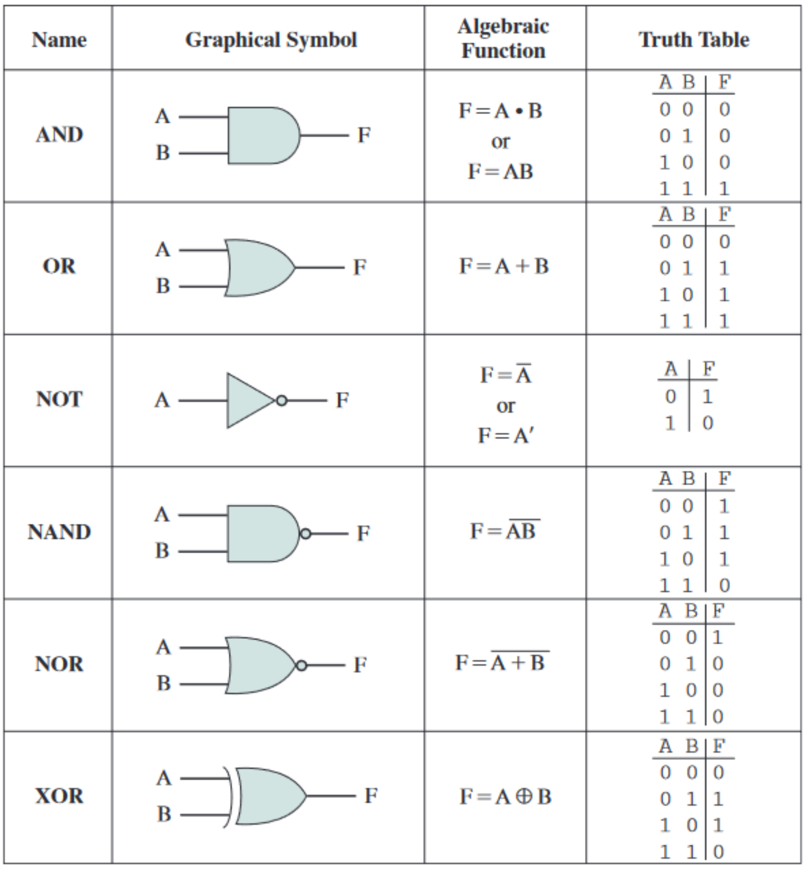
Boolean algebra are algebra based on only the values 1 or 0.
It consist of variables and the operations AND $(\cdot)$, OR $(+)$ and NOT $(\bar{b})$ in that precedence.
Another often usefull operators are XOR $(\oplus)$, NAND $(\overline{b \cdot a})$ and NOR $(\overline{a + b})$
Set operation may also 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, but any number of inputs is possible and some gates may have two outputs where one is negated.
When designing af 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 indentities, which can simplify the expressions



Name	Graphical Symbol	Algebraic Function	Truth Table
AND	A B F	$F = A \bullet B$ or $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR	A B F	$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT	A F	$F = \overline{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND	A B F	$F = \overline{AB}$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR	A B F	$F = \overline{A + B}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR	A B F	$F = A \oplus B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

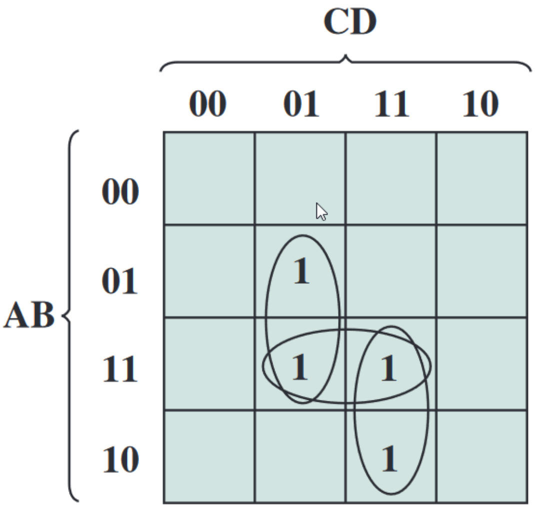
Figure 1: Figure of gates and their logical operation

- 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 posibilities are accounted for.
So for one side describing one variable there are 2 possibilites and a row which describes two variables there will be 4 possible outcomes.
For each row/column combination the function or truthtable 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.

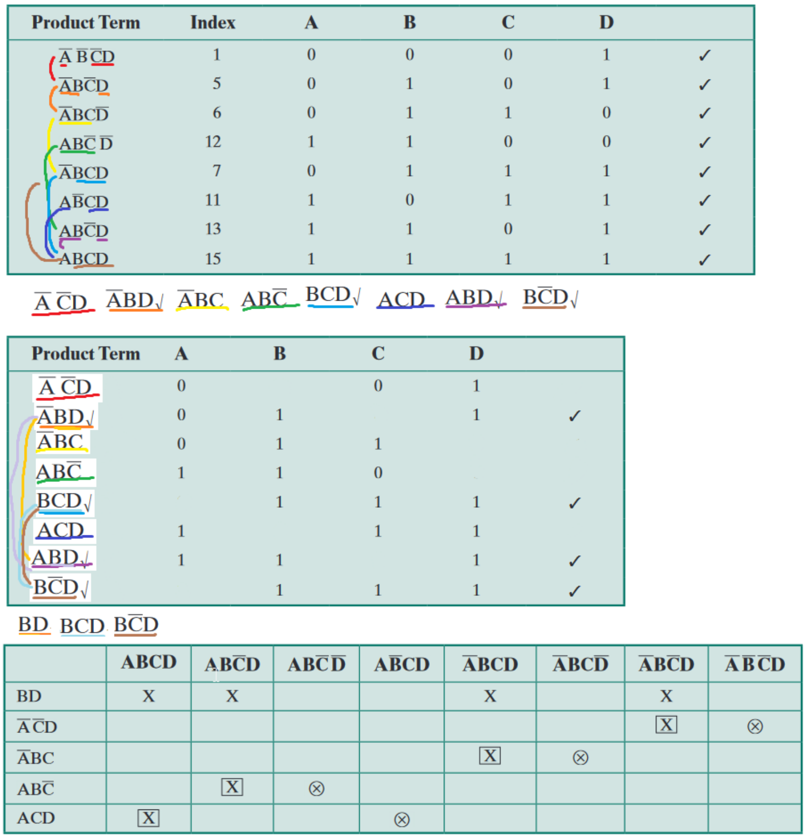


(b) $F = B\overline{C}D + ACD$

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 elment 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	
$\overline{A}\,\overline{B}\,\overline{C}D$	1	0	0	0	1	✓
$\overline{A}B\overline{C}D$	5	0	1	0	1	✓
$\overline{A}BC\overline{D}$	6	0	1	1	0	✓
$AB\overline{C}\,\overline{D}$	12	1	1	0	0	✓
$\overline{A}BCD$	7	0	1	1	1	✓
$A\overline{B}CD$	11	1	0	1	1	✓
$AB\overline{C}D$	13	1	1	0	1	✓
$ABCD$	15	1	1	1	1	✓

$\overline{A}\,\overline{C}D$ $\overline{A}BD$√ $\overline{A}BC$ $AB\overline{C}$ BCD√ ACD ABD√ $B\overline{C}D$√

Product Term	A	B	C	D	
$\overline{A}\,\overline{C}D$	0		0	1	
$\overline{A}BD$√	0	1		1	✓
$\overline{A}BC$	0	1	1		
$AB\overline{C}$	1	1	0		
BCD√		1	1	1	✓
ACD	1		1	1	
ABD√	1	1		1	✓
$B\overline{C}D$√		1	1	1	✓

BD BCD $B\overline{C}D$

	$ABCD$	$AB\overline{C}D$	$AB\overline{C}\,\overline{D}$	$A\overline{B}CD$	$\overline{A}BCD$	$\overline{A}BC\overline{D}$	$\overline{A}B\overline{C}D$	$\overline{A}\,\overline{B}\,\overline{C}D$
BD	X	X			X		X	
$\overline{A}\,\overline{C}D$							$\boxed{X}$	⊗
$\overline{A}BC$					$\boxed{X}$	⊗		
$AB\overline{C}$		$\boxed{X}$	⊗					
ACD	$\boxed{X}$			⊗				

Figure 3: Example of the method on
$F = ABCD + AB\overline{C}D + AB\overline{CD} + A\overline{B}CD + \overline{A}BCD + \overline{A}BC\overline{D} + \overline{A}B\overline{C}D + \overline{ABC}D$
Which result in the list $F = AB\overline{C} + ACD + \overline{A}BC + \overline{AC}D$

3.4 Circuits

3.4.1 Multiplex

Multiplex is a circuit of which a number of inputs label D0,D1,...,DN is wired to an output F.
To controle which input determine the F value, the required number of selection inputs are used called S1,S2...,SN.
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 usefull for writting 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 implmented 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 ouputs 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 flips flops can then be made in parrallel 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 assyncronous counter, which when incrementet 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 flips when the right bit is 1.

3.4.5 Programmable logic devices

PLD are general-purpose chips.
There are different types of PLD

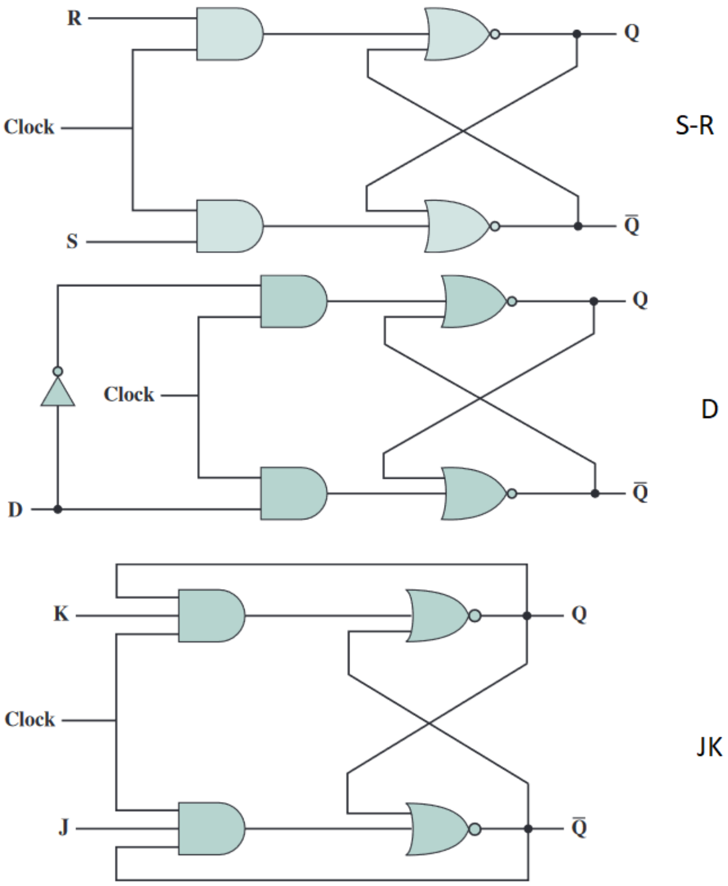


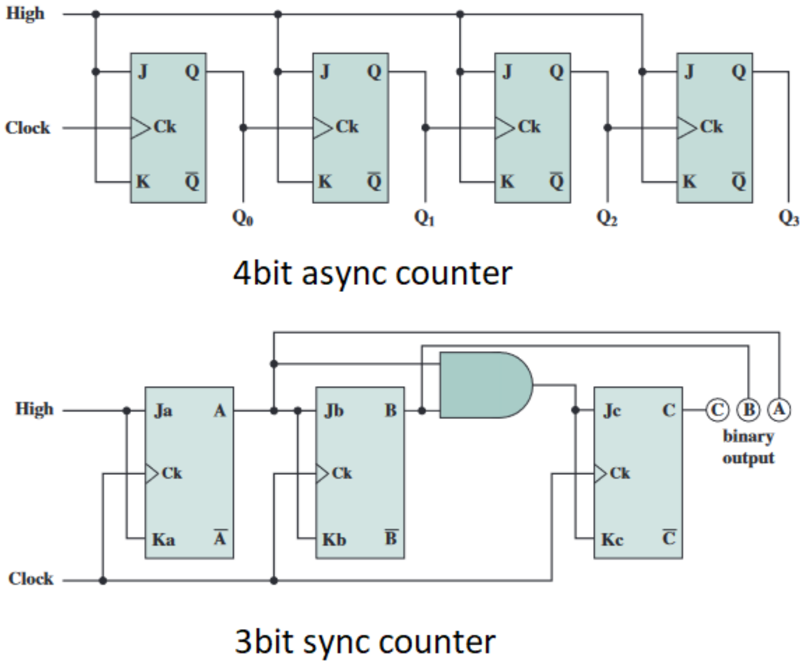
Figure 4: Different types of flip flops

- 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 og 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:



4bit async counter

3bit sync counter

Figure 5: Two implmentations 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 - Artihmetid 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 referenece - 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 avaliable
- Instruction format - Instruction lengthm number of addresses, size of varius field 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 flaoting point - here the first bit mean the sign(1 = negative) the nest 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, whcih 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, detemine if data is not cached and issur a command to the memory module.
- Arithmetic - Different kind of matematic 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 aswell as operation conversion between 8 bit and such
- I/O - Instructions for data movement in and out of the system
- System control - Privelege 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 specificly 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 parrallel on things like image processing where pixels are gathered in larger chunks.

5 Addressing modes and Formats

5.1 Adressing modes

Adressing modes are different methods of getting an address which can be send to the accumilator.
Most often a system implements two modes and the selected method is either in the opcode or in the memory field.

- Immediate - The address is in the instruction, needs little memory but has size of address is limited to operand
- Direct - The instruction contains a memory location which value is the address, memory location is limited to operand size.

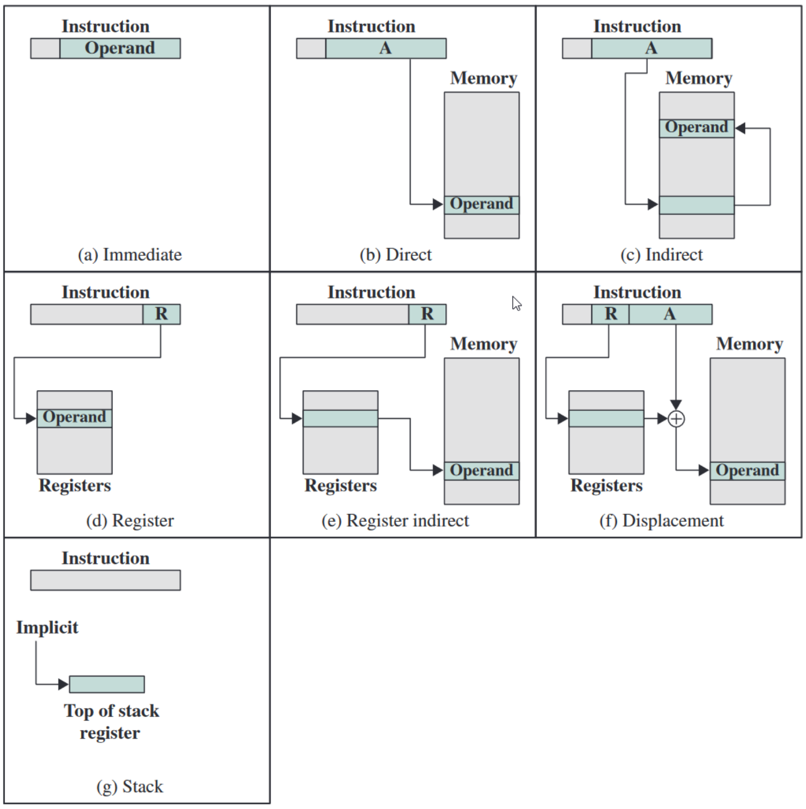


Figure 6: Illustrations of addressing modes

- Indirect - The instruction contains memory locations which contains memory location of which value is the address, first operand location is then low such it can contain a higher memory location
- Register - Same as direct but the operand refers to a register instead, registers are faster in case of reuse and requires a smaller address
- Register inderect - Same as indeirect but with registers
- Displacement - Operants are both a register address and a value which is added to the register value to find memory location containing address.
- Stack - Instead of instruction including a memory reference the stack is used to operate

Displacement can be used in different ways

- Relative - The register used is the program counter, such the memory location is relative to the current with the offset of the other operand.

- Base register - Uses the base registers value added with the other operand
- Indexing - Register contains offset and other operand is memory location, often used for loops, and autoindex may be enabled where it automaticly in a cycle increment.

For autoindex, the new index is saved in the memory location, this is either done before preindexing or after the indirection it is postindexing.

5.2 Addressing in x86

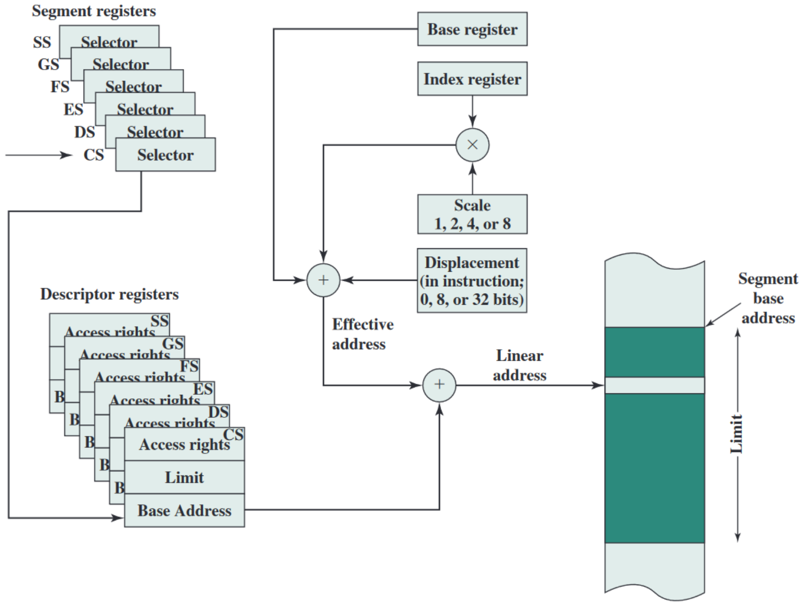


Figure 7: Addressing modes in x86 and how it is calculated

In x86 there are segment registers, these registers holds an index of the descriptor registers.
The descriptor registers hold 3 values: Access right to the data, how much data there is and where the first part of the data is located.
In addition there is a base register and index register.
For register operand mode - either a 32 bit register, 16 bit register or 8 bit reguster can be used for data tranfer, aruthmetic and logical instructions.
Displacement mode are not often used due to the long length up to 32 bits and are mostly used for global variables.
For the rest if the addressing modes the memory location is referenced by

the segment and the offset in the segment.
Displacement mode with base is used for local variables, index of arrays and large record pointers.

5.3 Addressing in ARM

On arm there are three alternatives to indexing due to no index register being a thing.
Offset - The instruction holds the offset as well as the register with the address.
Preindex and postindex are offset but with the writing to the register either pre or post indexing.

5.4 Instruction formats

The length of an instruction is determined by a lot of factors, the longer the easier to program but more space.
The length should be equal to memory-transfer or a multiple the length to ensure integral nnumber durin a fetch cycle.
The length should also be optimized to be shorter due to memory most often being a bottlenect in speed.
The instruction length should also be a multiple of 8 due to the character length, and how that relates to the word length such a word contain an integral number of characters.

The allocation of bits is also determined by multiple factors
The more opcodes the more readable code and often less code, but som opcodes may be determined by the operands also.
The amount of registers also affect the amount of bits required to describe a register. The ideal amount is found to be between 8 and 32.
These registers can also be in sets such they can be determined with smaller amount of bits ex 2 sets of 8 requires 3 bits of data.
The operands also need a good amount of data for addresses, not directly addresses but rather a big range for the displacement.
Variable length instruction are variable length in the bit allocation in instruction which solves some of the many problems but requires more complexity in the processor and multiple instruction may be fetched at once.

5.4.1 x86 instruction format

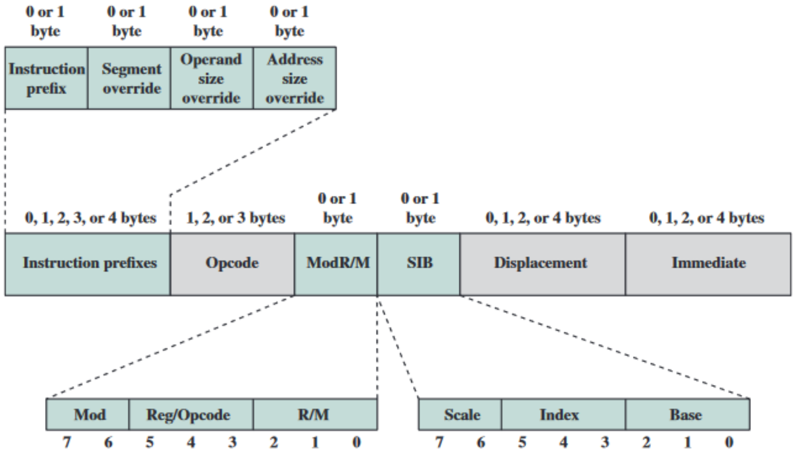


Figure 8: x86 instruction format and it lengths

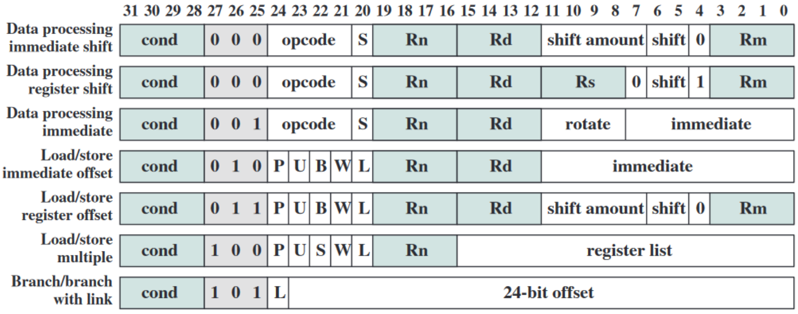
- **Prefixes**
- Instruction prefixes - There are two avaliable prefixes, LOCK which ensures exclusive use of shared memory in multiprocessor envirements, and repeat prefixes which can be on of 5
  - REP - repeats instruction until register CX is equal to zero
  - REPE/REPNE - repeats until value of ZF flag
  - REPZ/REPNZ - repeats until RCX, ECX or CV is equal 0
- Segment override - Overrides which segment register an instruction should use
- Operand size - Switches between 16 or 32 bits operands
- Address size - Switches between 16 or 32 bit address generation
- **Instruction**
- Opcode - 1 - 3 bytes of length may also include bits about: if data is byte- or fullsize, direction of data operation, immediate data field must be sign extended
- ModR/M - The location of the first operand (address mode or register) and the location of the second operand (a register) if required by the instruction. Or an extra bit for the opcode (opcode extension)

- SIB - If a specified address mode need more data the SIB contain: The scale field for scaled index (2 bits), index field (3 bits) for index register and base field (3 bits) for base register.

- Displacement - If displacement addressing mode is used, an 8-, 16-, or 32-bit signed integer displacement is specifed

- Immediate - Provides the value of an 8-, 16-, or 32-bit operand

5.4.2 ARM instruction format



S = For data processing instructions, signifies that the instruction updates the condition codes

S = For load/store multiple instructions, signifies whether instruction execution is restricted to supervisor mode

P, U, W = bits that distinguish among different types of addressing mode

B = Distinguishes between an unsigned byte (B==1) and a word (B==0) access

L = For load/store instructions, distinguishes between a Load (L==1) and a Store (L==0)

L = For branch instructions, determines whether a return address is stored in the link register

Figure 9: ARM instruction formats

- Immediate constants - 8 bit constant value which can be rorated by 4 bit to create constants

- Thumb instruction set - A subset of instruction which have been decreased to 16 bit instead of 32 bit. This is done by
  - Thumb instruction are unconditional, so the conditional field is not used and all arithmetic thumbs update the conditions flag, so not flag bits are needed
  - The limited amount of opcodes only require 2 bits and 3 bit type field
  - Thumb instructions only references registers r0 to r7 so only 3 bit is required

- Thumb-2 instruction set - This is the bridge between ARM and thumb instruction which makes it possible to combine both instruction for more compact and performed code

6 Assembly language concepts

Assember - compiler for assembly to object code
Linker - Combines one or more files containing object code into a single loadable file or executable code
Loader - Copies an executable into memory for execution
Object code - Step between assembly and executable code Symobilic program - A static somewhat like assembly language with static memory
Assembly uses symbolic addresses for data for non static references.
The downsides of writing in assembly instead og high level language

- development time takes much longer
- Reliability and security are lower due to no compiler which can warn or throw errors of bad or unsecure code
- Debugging and verifying take longer due to more code resulting in more places it can go wrong
- Maintainability is low due to the often spaghetti like structure of assembly
- Protability is only on same platform
- HLL language have access to use intrisc functions so assembly is not required for device drivers and other system code
- Compilers are so good now that it often harder to write better assembly than the compiler

But there are upsides

- A compiled assembly code can be verified
- To create a compiler assembly is required
- Embedded systems may not be able to have a compiler and therefore need all code in assembly

- Hardware drivers and system code are often easier to program with assembly due to hll's not having access to hardware or registers or so on
- Accessing instructions that are not accessible from hhl
- Code size are often smaller with self written assembly
- Writing in assembly makes it possible to optimize more in speed than a compilers general optimization

6.1 Assembly language elements

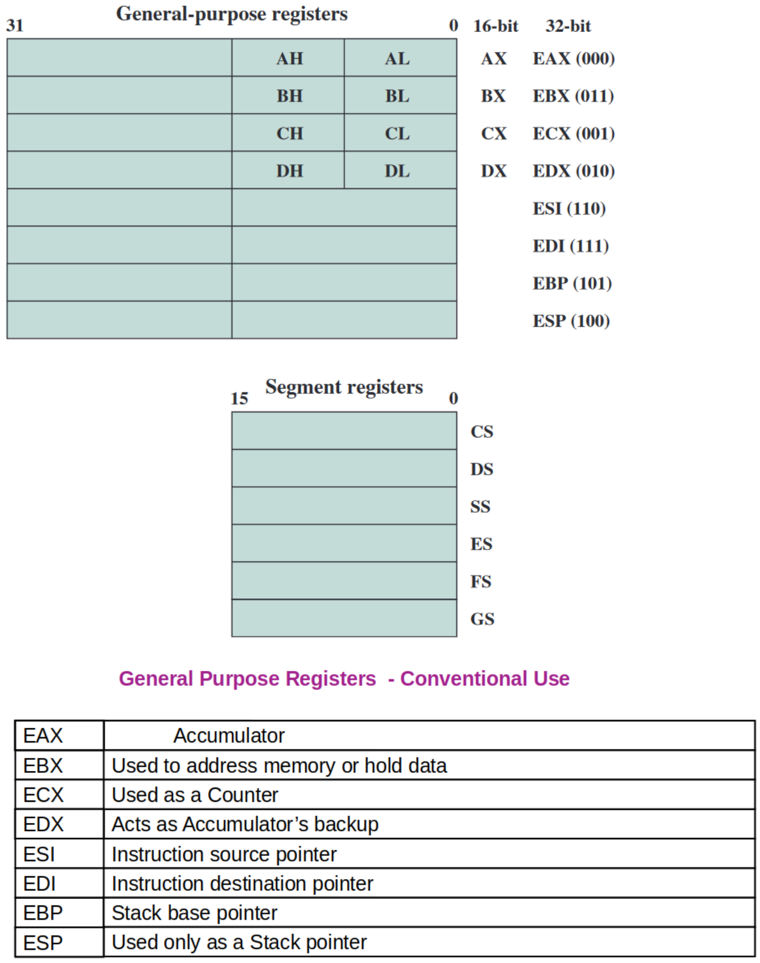
An assembly statement consist of: label, mnemonic, operand and comment

- Label (Optional) - Equivalent to address used for code references
- Mnemonic - Name of operand or function , symbolic name representing an opcode
- Operand(s) - zero or more operands may be given representing either a immediate value, a register value or a memory location
- Comment (Optional) - Text which are ignored by help the programmer in x86 it starts with a semicolon

When referencing data in operands there are different methods as discussed. For register addresses the register name is used. For immediate addressing uses an indication that the value is encoded. Examples are H for hexadecimal, B for binary and decimal has no suffix
Ex. 100B is read as a binary value
For direct addressing expressed as a dispalcement from the DS segment.

6.1.1 Pseudo-instructions

Pseudo instruction are not x86 machine instruction but are still placed in the instruction field.

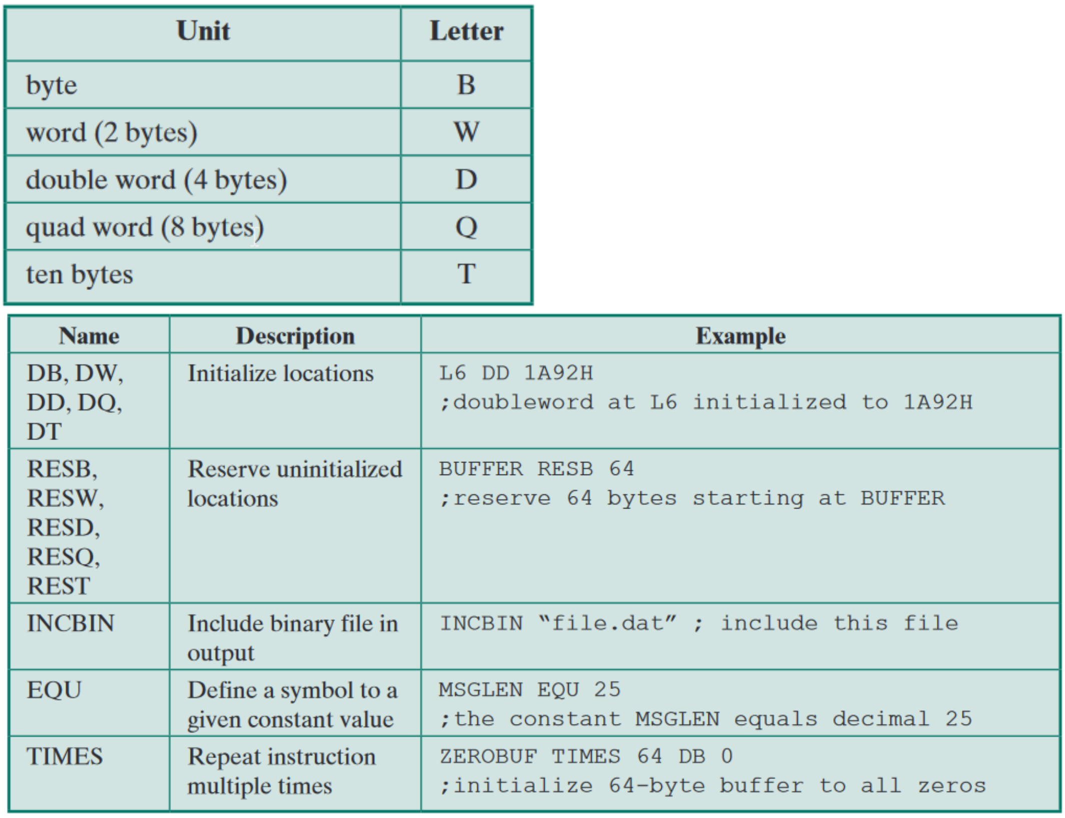


EAX	Accumulator
EBX	Used to address memory or hold data
ECX	Used as a Counter
EDX	Acts as Accumulator's backup
ESI	Instruction source pointer
EDI	Instruction destination pointer
EBP	Stack base pointer
ESP	Used only as a Stack pointer

Figure 10: x86 register names and their uses

6.1.2 Macro definitions

Macros are a subroutine of code, and can be used multiple times like a call instruction.
The main difference is macro expansion which actually inserts the macro at every instance to get rid of overhead time of instruction switching.
This will result in a larger code but more clean code.
Therefore a macro should only be a small bit of code.
Macros can both be defined on single lines as - %DEFINE A(X) = 1 + 8 + X and then be referede to as - MOV AX, A(8)
For a multiline macro
%MACRO ¡NAME¿ ¡number of params¿



Unit	Letter
byte	B
word (2 bytes)	W
double word (4 bytes)	D
quad word (8 bytes)	Q
ten bytes	T

Name	Description	Example

DB, DW, DD, DQ, DT	Initialize locations	`L6 DD 1A92H`
		`;doubleword at L6 initialized to 1A92H`

RESB, RESW, RESD, RESQ, REST	Reserve uninitialized locations	`BUFFER RESB 64`
		`;reserve 64 bytes starting at BUFFER`

INCBIN	Include binary file in output	`INCBIN “file.dat” ; include this file`

EQU	Define a symbol to a given constant value	`MSGLEN EQU 25`
		`;the constant MSGLEN equals decimal 25`

TIMES	Repeat instruction multiple times	`ZEROBUF TIMES 64 DB 0`
		`;initialize 64-byte buffer to all zeros`

Figure 11: Directives unit and letter and whtat the different pseudo instruction do

```
PUSH EBP;
SUB EBP, %1 ; first parameter
%ENDMACRO
```

6.2 Types of assemblers

- Coss-assembler - Runs on another computer host to then be transfered to the target machine
- Residen assembler - Host and target are the same
- Macroassembler - Allows the user to define sequences of instructions as macros
- Microassember - Used to write microprograms which define the instruction set for a microprogrammed computer
- Meta-assembler - Can handle multiple instruction sets
- One-pass assembler - Produces machine code from a single pass of the assembly code
- Two-pass assember - Makes two passes to produce machine code

Two pass works by first finding and creating a symbol table of all symbols and their given location counter (LC).
In the second pass the following is done

- Transte the mnemonic into binary upcode
- Use opcode the analyze the instruction
- Translate each operand to appropriate register or memory code
- Translate immediate value to binary
- Translate labels reference to LC
- Set any other bit given in the instruction

The assembler also has a zeroth pass where it reads all macros which are defined at the top, such it can expand on first pass.

For a one pass compiler the trouble is keeping reference while translating which is done by, when a new label is found the following is done

- Leaves the instruction operand field empty in the assmbled binary instruction
- The symbol used as an operand is entered in the symbol table and flaged as undefined
- The address of the operand field is added to a list of forward references associated with the symbol table entry

6.3 Loading and linking

Linker - Finds references to other code or data and links the references between object code (non linked machine code)
A linker which produces a single module from the main module and it references to other modules is called the linage editor
Load-time dynamic linking keeps all references to external modules and in run time when the external module is used it is loaded into main memory and the reference is updates.
This also makes the libraries updatable and be in files themselves, ex in windows they are in dynamical-link libaries (DLLs), but this can also lead to DLL hell where two executables expect two different version of the same DLL file.

Loader - loads the final machine code into main memory
There is 3 types of loading

- Absolute loading - Requires all modules are in the same loacation in memory, and references are absolute, this has many disadvantages with everything having to be in main memory at once and the absolute nature makes it near impossible to make correct references
- Relocatable loading - Every reference is relative to some point in the program, such that point location is just added to every other reference
- Dynamic run-time loading - To ensure that the program has not been moved since first loading, the references is first calculated with the offset when the reference is needed

7 Assembly language x86

A list of every instruction can be found here
The structure using at&t is mnemonic source, destination.

The registers start with % and litteral valeus start with $
For comments # is used.
For 64 bit instruction, the opcode ends in q

7.1 Directives

Directive are part of code to initialize data or write the code.
There are 3 types of sections .text, .bss and .data
.data is for storing data and .bss is for allocating data.
.text is for the code and global variables.
Other forms of directives are for storing or allocating data.
This can be done by: .space ¡numBytes¿ (Byte 8, Word 16, Doubleword 32, Quadword 64, Doulbe quadword 128)
Or with text and label: myString: .ascii ”Hello World!”

7.2 Starting the code

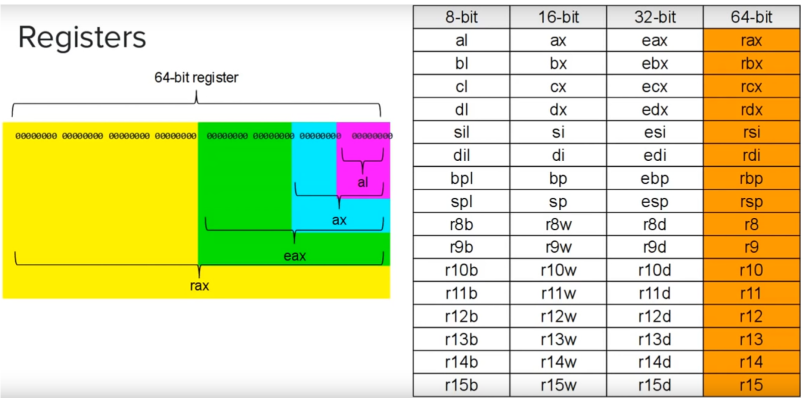
The assembler looks for the start of the code by:

```
section .text
   global _start
_start:
   ...
```

7.3 Registers

[h!] From the existing registers the same space is actually used.
The different registers and their space can be seen in the figure.
In some instructions two registers may be used for wider range indicated by reg1:reg2 ex: RDX:RAX

- RAX - accumilator for arithmetic operaions
- RBX - Base, holding a pointer to data
- RCX - Counter used in shift/rotate instructions and loops
- RDX - Data, used in arithmetic operations and I/O operations
- RSI - Source index, a pointer to data source in stream operations
- RDI - Destionation index, a pointer to data destination



8-bit	16-bit	32-bit	64-bit
al	ax	eax	rax
bl	bx	ebx	rbx
cl	cx	ecx	rcx
dl	dx	edx	rdx
sil	si	esi	rsi
dil	di	edi	rdi
bpl	bp	ebp	rbp
spl	sp	esp	rsp
r8b	r8w	r8d	r8
r9b	r9w	r9d	r9
r10b	r10w	r10d	r10
r11b	r11w	r11d	r11
r12b	r12w	r12d	r12
r13b	r13w	r13d	r13
r14b	r14w	r14d	r14
r15b	r15w	r15d	r15

Figure 12: The different registers in x86 assembly

- R8-R15 - New registers in 64 bit
- RSP - Stack pointer, points to top of stack
- RBP - Strack frame base, pointer to base of current stack frame
- RIP - Instruction pointer

7.4 System calls

To make system calls the call and its arguments are putted into the following registers.

- ID - rax
- 1 - rdi
- 2 - rsi
- 3 - rdx
- 4 - r10
- 5 - r8
- 6 - r9

Every system call and their ids can be found here.
When every value is set the instruction 'syscall' can be called.

To end a program a exit system call is made with argument 0 symbolising the error code.

```
section .text
   global _start
_start:
   movq $60, \%rax \#exit id is 60
   movq $0, \%rdi
   syscall
```

7.5 Jumps

First the instruction cmpq ¡arg1¿ ¡arg2¿ is ran to set the flag
Then one of the follwing condiiton jumps can be made

- je: $arg1\ ==\ arg2$
- jne: $arg1 \,!= \ arg2$
- jg: $arg1\ <\ arg2$
- jge: $arg1\ <=\ arg2$
- jl: $arg1\ >\ arg2$
- jle: $arg1\ >=\ arg2$

7.6 Stack and function calls

The stack goes from higher address to lower address.
So when a new thing is pushed on the stack the RSP is decreased by 8 for each stack.
When a function is called the stack is used in the following order to create a stack frame.

- Argument in reverse order
- Current instruction pointer
- Local variables created in function

The called function is responsible for popping off the local variables and instruction pointer.
The calling function has to pop the arguments.

For first 6 integer/pointer arguments are in registers
RDI, RSI, RDX, RCX, R8, and R9
If more arguments are needed to stack is used.
RAX is used for return value.
For saving old register values they may be pushed to stack before beginning to function call.
To call a function: call ¡label¿
And to return back from call: ret

8 Computer Arithmetic

The core of the computer is the arithmetic unit (ALU)
Representing positive or negative a fixed number a bits is used to represent the number and the right most bit is a sign bit.
This causes some drawbacks, mainly arithmetics are not as simple and checking for zero is also not as easy.

8.1 Sign-magnitude representation

The left most bit represent the sign if 1 then it is negative if 0 then positive.
Drawbacks are bad arithmetic and hard to check for zero.

8.2 Twos compliment

Twos compliment counter act this by positive must the first bit be 0.
Twos compliment of size n is defined such if the most significant bit is 1, 2^n is subtracted to make it more simple.
When negating twos compliment the most significant digit is flipped.
The two edge cases 0 negation and the largest number negation, here a carry has to be remembered an used.

8.2.1 Twos compliment arithmetic

Doing addition is the same as binary, but the two most significant bits are not added rather XOR'ed
For subtraction a negation is done and then an addition
The addition is therefore done by a single addition unit taking two registers, saving the result in one of them or a third register, and indicate a possible

overflow in a flag.

8.2.2 Twos compliment multiplication

For multiplication $n \times m$ bits, the n bits are run through for every itteration a shift right of m is done. If the bit itteration in n is 1, m is added to the summation. For multiplication of twos compliment, the multiplicand is in register M, multiplier in Q and two extra registers are needed A with start value of 0, and a single bit register we call Q_{-1} to represent it being to the right of itteration of Q.
Then Q is itterrated through and looking at both Q_i and Q_{i-1}.
If the two bits differ then, if Q_{i-1} is 1 then M is added to A, and if Q_{i-0} is 0 then M is subtracted from A.
If the two bits are the same, or an addition/subtraction is done, then A and Q is cyclic shifted to the right as one unit, and Q_{-1} is what would have shifted into -1
This is repeated n times where n is the length of the multiplier.
The result will be AQ.

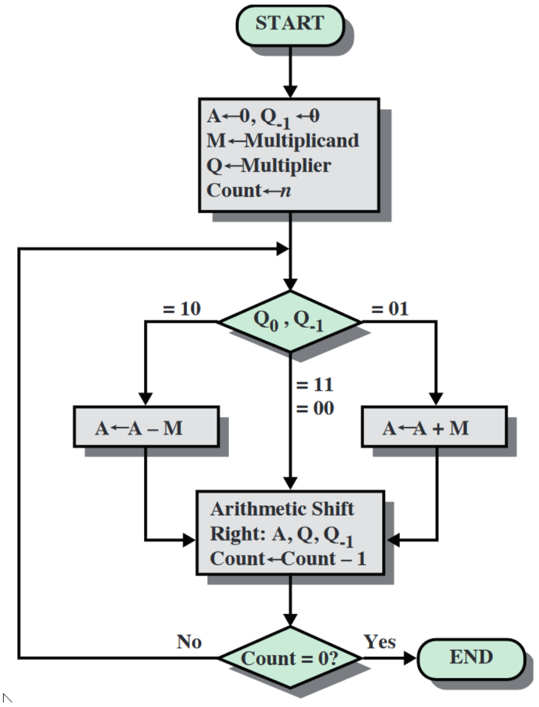


Figure 13: Twos compliment multiplication diagram

8.2.3 Twos compliment division

For division it differs not far, where M is the divisor, Q is the divident and a summation register A with initial value 0.
First A and Q is cyclic shifted left as one unit
Then M is subtracted to A.
If A is larger than zero (rightmost bit is 0) then set $Q_0 = 1$ otherwise set $Q_0 = 0$ and add M to A.
The remainder will be in A and the quotient is in Q.

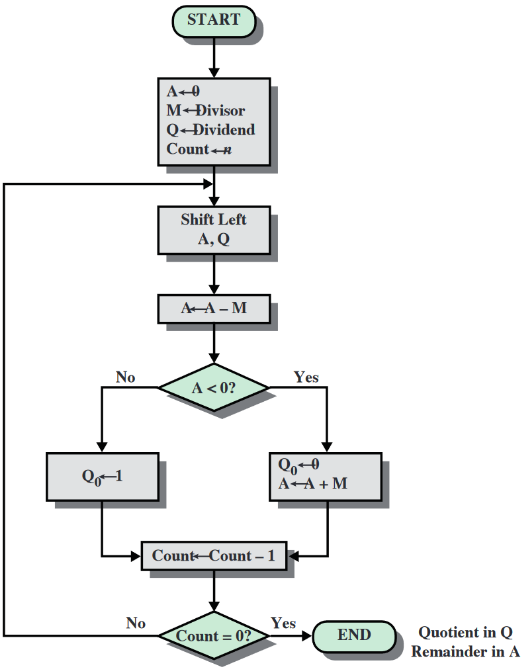
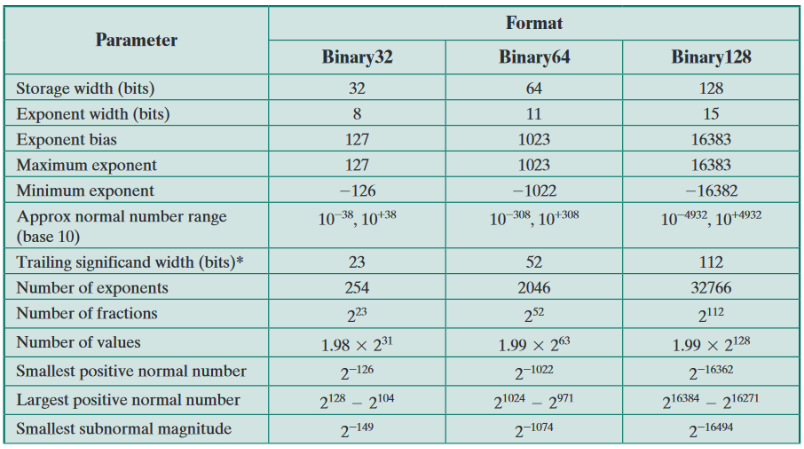


Figure 14: Twos compliment division diagram

8.3 Floating-point representation

For a typical 32 bit format of a floating point will be
1 bit for sign,
8 bits for exponent,
23 bits for mantissa
$sign 1.mantissa \cdot 2^{exponent}$
For 64 the exponent is 11, and 128 15bits
The exponent has a default bias being the negative value of half the range.
This giving the values: It can here be seen that the density of representable



Parameter	Format		
	Binary32	**Binary64**	**Binary128**
Storage width (bits)	32	64	128
Exponent width (bits)	8	11	15
Exponent bias	127	1023	16383
Maximum exponent	127	1023	16383
Minimum exponent	-126	-1022	-16382
Approx normal number range (base 10)	$10^{-38}, 10^{+38}$	$10^{-308}, 10^{+308}$	$10^{-4932}, 10^{+4932}$
Trailing significand width (bits)*	23	52	112
Number of exponents	254	2046	32766
Number of fractions	2^{23}	2^{52}	2^{112}
Number of values	1.98×2^{31}	1.99×2^{63}	1.99×2^{128}
Smallest positive normal number	2^{-126}	2^{-1022}	2^{-16362}
Largest positive normal number	$2^{128} - 2^{104}$	$2^{1024} - 2^{971}$	$2^{16384} - 2^{16271}$
Smallest subnormal magnitude	2^{-149}	2^{-1074}	2^{-16494}

Figure 15: Limits for the different floating point formats

numbers is lower on higher numbers.
When working with floating-point arithmetic one of the follwing may happend

- Exponent overflow
- Exponent underflow ending up being reported as 0
- Significand underflow/overflow ending up rounding such nothing really is changing

Subtraction and addition can be split into four groups

- Check for zeros - Change sign if subtraction, and check if one is equal to 0 such the result can be returned
- Align the significands - The lower number is shifted to the right and exponent is incremented, such the least significant bits are may lost
- Add or subtract the signifands - If overflow/underflow occur the exponent is either incremented or decremented
- Normalize the result - left shift until the left most digit is not zero and decrementing the exponent

Multiplication

- Check for zeros
- Exponents are added together

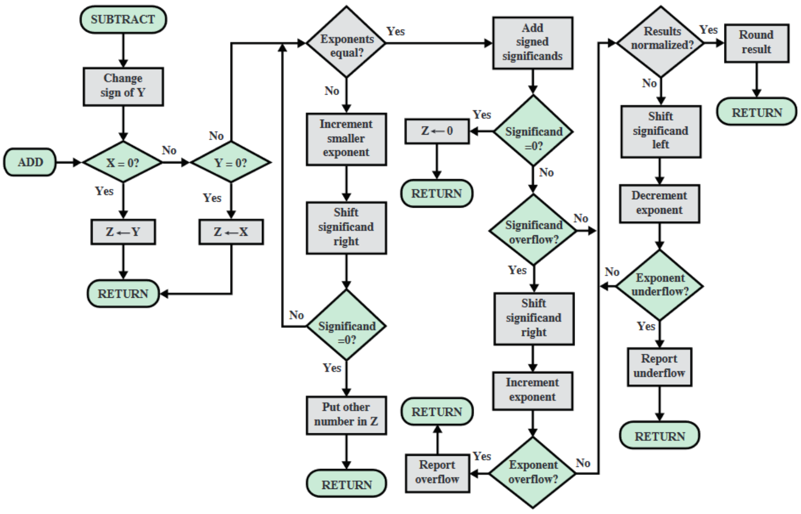


Figure 16: Floating point addition and subtraction diagram

- Significands are multiplied as normal integers
- Normalize the result

Pretty much the same for division, but instead exponents are subtracted and significands are divided as normal integers.
Guard bits are extra zeros added to the end of the significant, such at left shift there will be less lost of significant bits.
When the result is put into the given format rounding may be needed, there are 4 different approaches

- Round to nearest representable number, does require some extra statement for the exact between of two representable numbers, standard is towards even
- Round towards $+\infty$
- Round towards $-\infty$
- Round towards 0

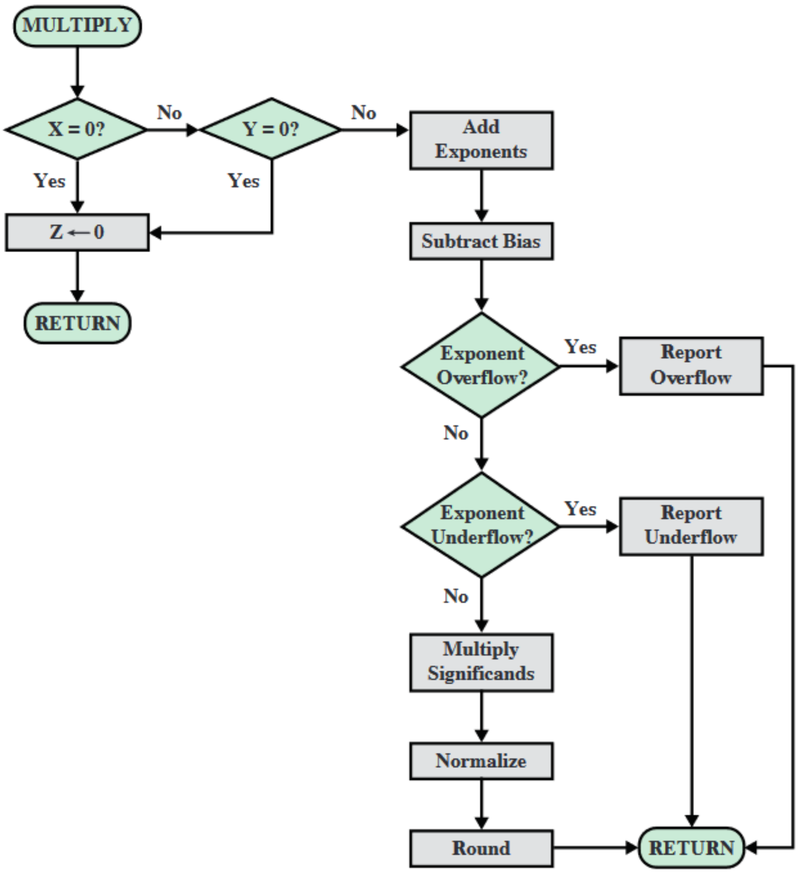


Figure 17: Floating point multiplication diagram

9 The Memory Hierarchy: Locality and Performance

9.1 Locality

Locality is the act of caching memory or instructions.
Locality comes in two types

- Temporal Locality - Recent used data which is saved in case of reuse
- Spacial Locality - Data around gathered data

Temporal locality is very usefull since many studies have found that many programs tends to use a small amount of data a lot and have access to a larger amount of data.

Likewise with spacial is the probability high espacialy for code since its most linear that surrounding data is usefull in the future.

9.2 Memory systems

Glossery

- Word - A unit of length depending on system intel x86 32 bit
- Addressable units - The amount of addresses
- Unit of transfer - Number of bits written or read to/from main memory
- Sequential access - Data is stored in linear form in records which is moved around in units
- Direct access - Memory is assigned to a physcial address
- Random access - Data is assigned to unique addressing mechanism
- Associative - Type of random access memory which compares address data
- Access time (latency) - Time to perforam a read or white operation
- Memory cycle time - Time to reset memory in RaM
- Transfer rate - the speed ofwhich data is transfered denoted $T_n = T_a + \frac{n}{R}$, T_n =aver time to read/write n bits, T_A=average acces time, n= number of bits, R = transfer rate

9.3 Multilevel Memory Hierarchy

The memory hierarchy is used to distribute data such larger data capacity has slower acces time, to compensate cost.
The hierarchy is groped as

- Registers
- On-chip cache
- Off-chip cache
- Main memory
- Flash memory

- Disk
- Off-line storage

on-chip cache is often implemented using SRAM and off-chip cache using eDRAM.
Cache unlike other memory is not visible to the programmer and is hardware controlled.
External nonvolatile memory is refered to as secondary,- or auxiliary memory
For implemneting a memory hiarachy the following must be supported

- Locality
- Inclusuin - Given data in one level the same data must be exist as a copy on all higher levels
- Coherence - If data is modified all copies must also be updated

10 Cache

Cache is used for temporary storage for faster access.
When data is not present in the cache a copy is read into the cache from which the CPU read the data.
The cache consts of blocks which is the minumum size of data transfer to the cache.
The blocks is contained in lines which can hold one block.
A tag is asociated with the line for addressing purposes.
Each line also include a controle bit indicating if the data in memory has been modified.
A transfer is often lees than a line which typically is 128 bytes and a block size being 2 bytes.
Cache is split into data and instructions, since it optimizes for old reuseable data.

10.1 Elements of cache design

High performance computing (HPC) deals with super computers which require a whole different approache to cache.
A program uses virtual addresses which is translated to physical by the memory management unit (MMU).
Cache can therefore save the virtual address or the physical.
By having the virtual a translation is not needed so it is faster, but has to

flushed for every new program and bits to the virtual address for different applications.
Since there is fewer cache lines than memory block a selection and organization has be done.
The most simple is direct mapping which simply takes the main memory block modulo with the avaliable cache lines which is the address.

10.2 Types of cache misses

- Compulsory Miss - Not present
- Conflict Miss - Data was overwritten
- Capacity Miss - The working set is larger than cache resulting in constant overwritting

10.3 Associative mapping

Associative mapping allows a physcial address to be placed in any line of cache.
This therefore requires every line the be checked for data in cache.
For the SRAM is used which is more expensive and contain less storage, but is able to in parralle in one clock cycle compare each line.
If the Main memory size is $= 2^k$ then k is the physical memory bits.
The block Size $= 2^b$ and b is the block offset.
The tag bits is then P.A. bits - Block offset.

10.4 Set Association mapping

This is a medium between associativ and direct mapping.
By dividing the cache into sets which is directly mapped to memory from which the sets are associative.
Calculating in this can be done as:
Line numbers: lines
MM address: x bit system
Block size
offset = log(blocksize)t
set number bits = index = log(line numbers)/(log(number of sets associated))
tag = log(MM) - offset - index
This then constructs the address with the same address size as the rest of the system but in the form

Tag— Index — Offset
So the address the first bits will be the tag, the next number of bits will be the index (i.e. the set it associated with) and the last bits will be the offset in the data. The index will then be at a set in which the cpu can search the the tag.

10.5 Replacement algorithms

Associative and set associative require a replacement algorithm for choosen which cache to be overwritten.
Least recently used (LRU) keeps a list of most recently used cache, when cache is used it is moved on top.
When overwritting something the bottom is used.
First in first out (FIFO) keeps a list of the longest staying cache and overwrites the oldest.
Least frequently used (LFU) uses a list of counter of use and replaces the least used cache.

10.6 Write Policy

When writing cache back to memory there are two cases
The memory has not been modified
The memory has been modified.
Memory may have been overwritten by the I/O or other caches.
Write through every update in cache is updated in memory which requires a lot of memory traffic.
Write back requires every write to happen in cache, when something is updated a cache a dirty bit is set from which memory can be updated
In case of write miss there are to approaches

- Write allocate - The block containing the word to be written is fetched from memory to cache to then write
- No write allocate - The block containing the word is written directly in memory

No write allocate is most often used with write through, but the amount of moving proves inefficient.
Write back and write allocate is most used, due writting to both cache and memory due to in a miss it will hit cache next time and setting the dirty bit for the block

In case of multiple devices with cache a cache coherency is needed, approaches can be

- Bus watching with write through - Each cache controller check memory of cached data and if updated invalidates local cache
- Hardware transparent - hardware is used to update modified memory in all caches
- Noncachable memory - shared memory is not cachable

10.7 Line size

The larger the line size the higher chance is relevant adjesant data is fetched. But larger lines overwrittes old cache and less blocks can be in cache at once. For larger lines the adjesant words becomes less likely to be used.
An optimum is found of 8 ti 64 bytes.

10.8 Inclusion policy

Inclusive policy dictates that data in one cache is guarenteed to be in all lower levels of cache.
This simplifies searching since the can be done in smallest fastest caches first and moving up.
Exclusive policy dictates that cache is guaranteed to not be in lower levels of cache, this is done to not waste space but requires more intense searching.
Noninclusive policy dictates that nothign is guarteed about placement of cache.

10.9 Cache timing models

t_{ct} = compare time of tag address and tag value in cache
t_{rl} =time to read a line from cache to retrieve data block in cache
t_{xb} = time needed to transmit byte or word to the processor
t_{hit} = time spend on cache level in case of hit
t_{miss} = time spend in cache in case of miss
Direct mapped cache
$t_{hit} = t_{rl} + t_{xb}$
$t_{miss} = t_{rl} + t_{et}$
Associative set cache
$t_{hit} = t_{rl} + t_{xb} + (1 - F_p)t_{et}$ where F_p is the fraction of time that the way preduction succeeds

$t_{miss} = t_{rl} + t_{et}$
For the associate cache it is the same as set but $t_{miss} = t_{rl} + t_{et}$

10.10 Performance modeling of multilevel memory hierarchy

For finding the average time to acces an item it can be expressed as:

$$T_s = H \cdot T_1 + (1 - H) \times (T_1 + T_2)$$

$$T_s = T_1 + (1 - H) \cdot T_2$$

T_1 = Acces time of fastest memory
T_2 = Access time to second level of memory
H = hit ratio

$$\frac{T_1}{T_s} = \frac{1}{1 + (1 - H)\frac{T_2}{T_1)}}$$

$\frac{T_1}{T_s}$ = Access efficiency.

11 Internal Memory

11.1 Semiconductor main memory

When talking about memory it is refered to as RAM (Random Access Memory) RAM comes in two types DRAM and SRAM
DRAM (Dynamic) uses capacitors to store charge, the charge can then be interpretted as 1 if charge is above a threshold.
The memory cell uses a transistor such two lines comes in a bit line and address line, the address line starts the circuit, if bit line is off it can be charged by the capacitor,
If the bit line is one the capacitor will be charged.
The dynamic name comes from the need to recharge due to the slow lost of charge in the capacitors.
SRAM (Static) uses flip flops to store memory and therefore is digital instead of analog
DRAM is cheaper but SRAM is faster and therefore used for cache memory.
ROM (Read Only Memory) has to advantage of being hard coded such it is nonvolatile and fast.

PROM is the programmable version but more expensive.
Read Most Memory is for memory which is most read version of this includes

- EPROM - Erasable Programmable Read Only Memory is a programmable ROM which can be deleted using UV lighting
- EEPROM - Electrical EPROM can be erased with electricity insetead of UV and is faster
- Flash memory - Is the fastest EPROM and provides erasabillity down to chunks but not bytes and has high density

The architecture of the memory cell is controlled by a chip.
The architecture decissions may include how big chunks the memory cell should be ex 8 Mbit chip organized in 1M x 8 memory cells.
This chips uses a multiplexer to convert the numbering of the cell into an electrical signal to the cell.
The chip then has input to wether a read or write should be performed and the output.
The architecture can therefore also chain multiple chips together for larger amount of memory but keeping clustered memory cells.
eDRAM is memory between the memory and chip often named L4 cache.

11.1.1 Error correction

For error correction parity bits are used.
These are extra bits used to ensure the stored bits are correct.
When writing or reading to memory, is the memory inserted into a function.
This function generates check bits in a given way.
When reading the checkbits and the memory is gathered, the memory is once again inserted into the function and the new checkbits is XOR'ed with the old creating the syndrome.
If an error have occoured to syndroms value will be equal to the bit which needs to be flipped.
This is an example of single-error-correcting (SEC) code.
Most often does semiconductor have both SEC and double-error-correction code which requirest a bit more.

11.1.2 Synchronous DRAM

Normal DRAM when getting request for data will the CPU have to check up if it has been delivered.

Using synchronous the CPU can instruct on which next cycle the memory should be avaliable.
SDRAM also makes use of multiple bank (set of chips) to perform parrallel data management.
SDRAM also allows certain length of burst of data to be written into the bus. Therefore making SDRAM fast at larger chunks of data.

11.1.3 DDR DRAM

DDR (Double Data Rate) is a spin on the SDRAM such data is gathered both on up and down clock cycles.
DDR also uses a higher clock rate for the BUS to increase transfer rate.
DDR also have a prefetch buffer on the chip, the memory buffer can get both buffer and memory on 1 cycle therefore making it possible for the BUS having the double clock rate.

12 External Memory

12.1 Magnetic Disk

Magnetic disk consist of a recording medium.
This medium is read by a the head consisting of magnetoresistive (MR) sensor or written to by a magnetic coil.
The medium is split up into tracks at different radius and these teacks contain sectors at variable length.
To compensate for different speed at the inner track and outside track the bits are written with different intervals, this therefore result in the outer track stores the same data size the same as inner track.
Therefore making the density the limit of data.
Multizone recording is a way to divide tracks into zones which have roughtly the same data density. This allows the data density to be higher the outer the tracks is..
A sector of data contains

- Gap
- Sync - Indicate start of sector and include timing alignment
- Address mark - Sector data including number, location and status

- Data
- ECC - error correction code

The format is in two ways the old 512 bytes where 50 is ECC and 15 is GAP, SYNC and Address mark.
The new format is 4k bytes, 100 ECC and 15 GAP, SYNC and Address mark.
Different types of mangetic disk

- Fixed head disk - A head for each track
- Movable head disk - Head moves inbetween tracks
- Nonremovable - Stationary disk unline portable versions
- Double sided - Magnetic coding on both sides of the medium
- Multiple platters - Stacked mediums and reader for more data
- Types of head
  - Air gap - Where air is present between reader and medium
  - No gap - The medium is more robust and allows reader to be on medium
  - Winchester - A foil rest on the medium when still and in spin the air pressur lift the foil

12.1.1 Performance

$$t_B = t_S + t_L + t_T$$

t_B - bloc access time
t_S - Seek time, head to track
t_L - Latency time, time for sector to reach head
$t_L = \frac{1}{2r}$
r - Rotation speed in RPS t_T - transfer time from memory to BUS
$t_T = \frac{b}{rN}$
b - Number of bytes transfered N - Number of bytes on track Rotational positional sensing (RPS) is a server feature which tries to gain IO access when the data is about to read, if not avaliable it will take a rotation and try again.

12.2 RAID

Redundant Array of Independent Disks comes in seben level 0 - 6
RAID uses a set of physical drives and creates logical drive.
The data is then distributed across the devices called stripping.
RAID also provides redundancy method for drive failure in some levels.
The RAID then also give the ability of faster data trasnfer through multiple IO's

0. Level - The data is striped out on all disk without any redundancy. The data is striped such continous data is spread out as much, such a single chunk can be split up into every drive.
1. Level - The data has a full dublicate, makes it two times the speed to get data, relative fast write since it depends only on slow drive, easy backup, lots of required capacity
2. Level - The data is striped down to byte level, gives fast read and error correction and uses hamming code for parity, require log x extra disk where x is number is disk.
3. Level - The data is striped down to bytes, parity is on a single drive where pairty bit of the same strip place, fast read due to byte level strip
4. Level - The data is striped in larger chunks, parity is depends on strips therefore making write time slower, for high I/O request fast but not big transfer
5. Level - Like level 4 but larger strip to prevent I/O bottle neck in RAID 4
6. Level - Like Level 5 but another disk is also used with another pairty method such two drives can fail

12.3 Solid State Drives

SSD make use of the NAND flash.
The SSD has:

- Higher input/output operations per second (OPS)
- More durability to physical factors

- Longer lifespan due to no moving parts
- Lower power consumption
- Lower access time and latency rate

The SSD also contain beside the flash

- Controller for interface and firmware execution
- Addressing
- Data buffer/cache
- Error correction logic and detection

The downside of the SSD is scattering problem. When data is scattered into multiple cell as consequence of filling up cell the write and read time will slow down.
The cell is also typically limited around 100,000 writes.

12.4 Optical memory

CDs and CD-ROMs are optical disk used for storing data.
With techniques like the HDD but with optical lasers.
Uses variable rotation speed, with a laster at a constant linear velocity.
The optical disk is engraved such at flat level represent 0 and changes in level represent 1.
The data is split into block of

- Sync - byte of 0, 10 bytes of 1 and a byte of 0
- Header - represent the following data and if error correction is at place
- Data
- Auxiliary - either extra data or error correction according to header

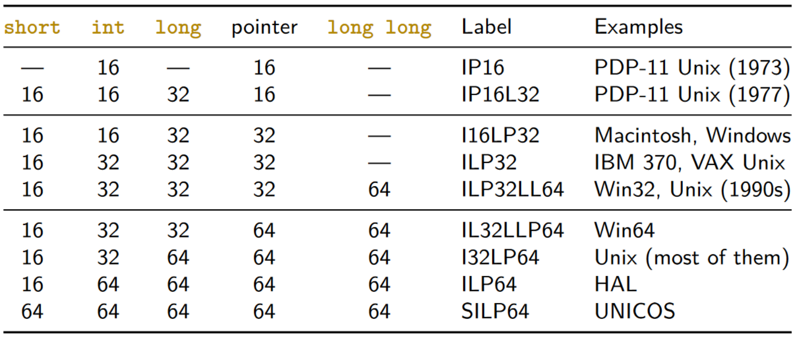
It has the advantage of cheap mass production and ability of storage of data and saving of data as backup.
CD-Recordable (CD-R) is a CD with a consumer friendly write
CD-Rewriteable (CD-RW) can be written to multiple times using a phase changeable crystal, with a limited amount of erase cycles
Digital Versatile Disk (DVD) is a more compact solution to CDs with capacity up to 4.7 GB
Blueray used like DVD a finer laser resulting in more density up to 25 GB.

13 C programming language

C is a performance constrained language which focuse on minimzing runtime overhed.
It is in direct controle of memory and compiles to native code.
Was designed for implementing UNIX
C has no support for

- Objects
- Exceptions
- Function overloading
- range check on arrays
- Garbage collection
- Limited type safety

To combat no run time error sanitizers are used. These run efore compilling checking for essentially run time errors.
Compiling C is done using gcc, this can be to object files using -c or directly to executable
To get warnings -Wall and for more warning -Wextra.
For debug -g is used.
So a good compile command should be:
gcc -Wall -Wextra -g -fsanitize=address -fsanitize=leak -fsanitize=undefined
Valgrind is an alternative to sanitizers which runs the program in a virtual a machine using:
valgrind -leak-chek=fall -track-origins=yes ./myProg arg1 arg2
The C program uses a main in the form int main(int argc, char **argv) or with just void.
An ampty parameter means the function takes any number of parameters.
Lines starting with # is used for preproccesing such as inclusions or macros
Macros simply takes blindly and copies the macro into the wanted places.
Macros are made as: #define PI 3.14 and undefined by #undef PI
Preprocessing can also use condictions like #ifdef DEBUG where -DEBUG is an argument to gcc
For negative variables signed versions is used



short	int	long	pointer	long long	Label	Examples
—	16	—	16	—	IP16	PDP-11 Unix (1973)
16	16	32	16	—	IP16L32	PDP-11 Unix (1977)
16	16	32	32	—	I16LP32	Macintosh, Windows
16	32	32	32	—	ILP32	IBM 370, VAX Unix
16	32	32	32	64	ILP32LL64	Win32, Unix (1990s)
16	32	32	64	64	IL32LLP64	Win64
16	32	64	64	64	I32LP64	Unix (most of them)
16	64	64	64	64	ILP64	HAL
64	64	64	64	64	SILP64	UNICOS

For error handling goto are often used as exceptions.
Arrays are initialed by: int arr[10] = {1,2,3} where the rest is just zeros.
Pointers are made by: *p = &i where i may be i = 42
Using **p = &p wil then be the pointer to the value of i ie 42
Additions to pointers is allowed and good use for array pointers.
Strings are made by char str[] = "Hello"

13.1 IO

getchar and putchar for single character input output to std
printf writes string to stdout including formatting.
scanf is the read equivalent
s version of printf and scanf does not include length check.

13.2 Memory allocation

Effective type is the type which the object relies on.
malloc is used for allocation space, with no effective type until wirtten, and returns a pointer to start of allocated space.
sizeof returns the size of an object type, ex sizeof(int)
free function free up the space after use and takes a pointer to the space which the free up.
calloc allocates space and fille with zeros
realloc is reallocating memory from one place to new place. The old pointer should never be used.

14 I/O

I/O modules contains logic for communicating between peripherals and system bus.
The module is reqired handling peripherials data format and mismatch between data transfer rate of processor and peripherals
Peripherals is split up into three categories: Human readable, machine readeable, and communication for remote devices.
Control logic associated with the I/O module controls peripherals in response to direction from IO module.
Transducer convert electrical signal to digital input, often with a buffer most often 8 to 16 bits or larger if block oriented.
The functions of the IO modules comes in categories

- Control and timing - coordinate flow of traffic between internal and external ressources
- Processor communication - Command decoding from processor to peripherals, data transfer between bus and peripherals, status reporting of devices, address recognition of peripherals
- Device communication - Commands, status information and data
- Data buffering - buffer data in case of different speeds
- Error detection - Error detection of mechinal and electrical malfunctions and error detection code

Three techniques are possible for I/O operations

- Programmed I/O - The program gets full control of IO, the processor must wait for next execution with IO and check when status is open for when finished
- Interrupt-driven I/O - Uses programmed I/O but interrupts the processor when done, such it can handle the data
- Direct memory access (DMA) - Uses interrupt driven IO but is able to transfer data to main memory itself

14.1 Programemd I/O

IO commands consist of which IO module, which peripheral and what command.
The commands is classified as

- Control - activate peripheral and issue a command
- Test - Test condition of IO module and peripherals
- Read - Obtain data from peripheral and place in internal buffer
- Write - Take data from system bus and transmit to peripheral

For addressing a peripheral there are two ways, memory-mapped I/O uses a single address space for memory location and IO devices, such peripherals is treated as memory locations.
The other is isolated I/O uses a single read line and single write line on bus, such it specifies wether a memory location or peripheral is being addressed.

14.2 Interrupt-driven I/O

The main idea is the IO module will interrupt the processor once done When the IO module completes an operation the follwoing steps is done.

1. I/O device issue interrupt singal to processor
2. Processor finishes what currently is going on and responds
3. Processor test for interrupt to find module and issue an acknowledgement signal to remove interrupt signal
4. Prepare to work with IO module by saving next instruction in the program status word (PSW), and registers onto the system control stack
5. Next instruction for the processor is set to the interrupt handling program
6. The interrupt is then handled where the whole IO is handled
7. Registers and PSW instruction is retrieved

For the processor to find the IO module which raised an interrupt there are different design approaches
Multiple interupt lines between processor and IO modules, but is impractical to dedicate multiple bus lines.
Software poll essentially ask every module until it find the correct one, but it is slow.
Daisy chain is hardware poll which chain every IO module and an ack is sent to all, from which it will be answered by the correct one, from which it can

be identified called a vector.
Bus arbitration also use vectored interrupts as in daisy chain which uses the bus to place the vector.

14.3 Direct Memory Access

The problem with Interrupt and programmed IO is the limited speed of which the processor can check for interrupt and the processor being tied up on IO transfers.
The DMA module is a moduel which connects to the IO and handles IO processes.
The DMA then only uses the bus when data has to be transfered and then steals cycles from the processor to use the system bus.
The DMA is commanded by sending a read or write controle line, the address of the wanted IO device, starting point of read / write, and the amount to write / read.
The DMA module is then able to handle memory and IO directly without the processor, and once done sends an interupt signal to the processor.
The DMA module can be connected to IO via the system bus, the IO can be connected directly to the DMA module or the IO is on a bus between the DMA and other IOs.

14.4 Direct cache access

Problem with DMA is it is too slow for some speeds today.
DCA uses the last level cache of the processor instead of main memory.
This can be seen when using networking at high speeds
The multiple unwrapping of protocols would result in many back and forward interaction between cache and main memory.

14.5 I/O channels and processors

An IO channel is an extension to DMA which make it able to itself execute IO instruction, such the CPU does not do any IO processing.
Instruction is stored in main memory and executed by a special purpose processor for the IO channel.
The instruction then include the needed infromation to perform the IO action.

There are two common types of IO channels.
Selector channel controls multiple high-speed deivces at one at a time.
A multiplexor channel works with a larger set of device at the same time with slower data throughput but can chain reads from different devices.