Floating point conversion (–1)sign\_bit × 1.significand\_bits × 2exponent\_bits – bias Convert decimal to binary, then convert to binary sci. notation. Places moved becomes exponent. Exponent = exponentbias – bias. Bias = 127 |Sign bit|8 exponent bits| significand (numbers to the right of decimal after sci. notation) -> break into 4 sections each and convert to hex. FPU – eight 80 bit data registers, ST(5), ST(6), ST(7) <-push ST(0) top of stack pop->ST(1), ST(2), ST(3), ST(4). 31h underlying integer for Q5.2. Convert 31h to binary. 2 most right then radix. Then 5 past radix. Convert to decimal. MASM DATA TYPE USED TO DEFINE SINGLE-PRECISION FLOATING-POINT NUMBERS, REAL4 – DOUBLE IS REAL8. NOTE: double precision conversion has 1 sign bit, 11 exponent bits, 52 significand, 64 overall, and 1023 bias.

FINIT – Initialize the FPU (called at beginning of main) FLD – load value from memory, push onto ST(0)

FST – copy (store) value from ST(0) into memory FSTP – store, then pop value off FPU stack

FADD, FSUB, FMUL, FDIV computes ST(1) op ST(0), pop both, push result. FILD – convert an SDWORD to floating point and push to ST(0)

FIST – round ST(0) to int and cop into memory FISTP – store then pop the value off the FPU stack

FCOMI ST(0), ST(i) – compare ST(0) to ST(i), copying flags to CPU, sets Z C P flags. FCOMIP pops afterward. (ffree st(0) then fincstp, identical)

Compare floating point (FP) by n2 – n1 < e for some small e.

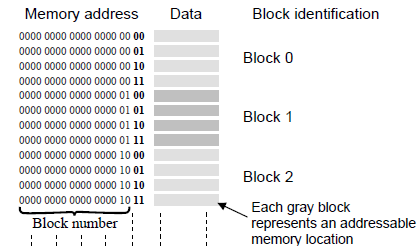
Heap Memory & Memory Management –Allocate heap memory: 1. call getProcessHeap (returns handle IDing it). 2. Push num of bytes to allocate, push 0, push handle, call HeapAlloc. Allocates mem then returns mem address in eax, 0 if error for getProcessHeap & HeapAlloc. To free heap, call getProcessHeap. exit macro is short for push 0, call ExitProcess.

2D Arrays - row major order – left to right. Column major order read top to bottom. Ex. Of base-index operand, mov ax,[ebx+esi], ex. Of base-index-displacement, mov eax,tableD[ebx + esi\*TYPE tableD].

RA mode ROM BIOS starts at F0000h, video mem at A0000h. BIOS provides partial mem map. RA mode, not for multitask. 20 bit mem addr, 00000h to FFFFFh, but 16 bit registers; only 1 MB of mem can be addressed. MSDOS – RAmode. Segment mem fixes register issue. Seg reg holds 16 bit seg value, 16 bit gen purpose reg holds 16 bit offset. Seg regs are CS, DS, ES, FS, GS. Actual mem addr is segment x 10h + offset. 08F1:0100 corresponds to 09010h. Protected Mode designed for multitasking, 32 bit mem addresses, 4GB of mem can be addressed. Process ea has own mem, cannot access other’s. Use flag segment. model. 32 bit addresses correspond to the linear address (offset) but all start at 00405000h, virtual memory.

Storage pyramid: <increased capacity>|Long-term (HDD) -> Main Mem (DRAM) -> cache RAM (SRAM)-> registers|<Faster> Stuff close together in mem, principle of locality; instructs executed in a short period, close in mem, data accessed in short period, close in mem. Temporal locality – if we use now, likely to use again soon. Spatial locality – if we use this now, need others nearby soon. Memory hierarchy takes advantage of storage techs to improve performance. Uses cache to be faster - Processor <-> SRAM <-> DRAM, when proc needs data, check cache, if cache hit use it, if cache miss, copy it; also copy from nearby locations (spatial). Percent of accesses in hits is hit rate/ratio. Cache Blocks and Lines divide main mem into blocks of fixed size, e.g. 64 bytes w/ each mem address divisible by 64. Cache stores copies of blocks when used. Blocks inside cache are called cache lines. If any byte in block access, copy whole block to cache. Upper bits of all mem addr within block are the same, lower 2 are different. Upper bits that are the same are the block number. Block number is the mem addr divided by block size. I.e.

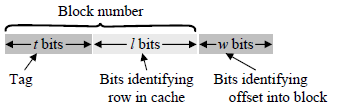
Suppose you have a 1GB(230-byte) addr space divided into 8-byte blocks (mem addresses are 30 bits).

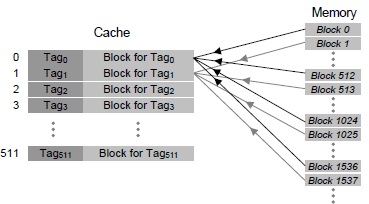


How many blocks in addr space? – 230/8 = 230/23 = 227 = 134,217,728

How would mem addr be divided to ID block and offset? a29 a28 … a3 (Bits IDing block)| a2 a1 a0 (Bits ID offset)

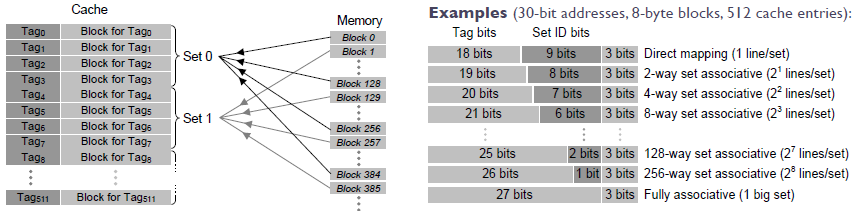
Cache questions: block placement/mapping function – direct mapped (Block number mod num of blocks to find slot), Fully associative (stored any slot), set associative (cache entries grouped into sets, block num determines set block will be stored in, but can be stored in any slot in set).

Direct Mapped – Number rows in cache from 0, store (see above), if there are 2x (x = l, pic) blocks, the lowest x block num = row num to store in. Tag = block number / num of rows.



Upper bits as tag to ID which block is stored in this row. Block IDing -decompose mem addr (see pic), check row of cache, does tag match? Yes – in cache, no – not in cache.

Fully Associative – same as pic above, combine l bits with tag to just have tag (whole block number).



Set Associative – if n entries in set, cache is n-way set associative. Block num determines which set to be stored in. Decompose as pic above. Each set is a small associative cache. (PIC TO RIGHT FOR SET ASSOCIATIVE)

Replacement Policy – choose alg. Least Recently Used (LRU) most common – whichever not read by processor in long time, FIFO, Least Frequently Used (since loaded in cache), Random.

Cache Write Policy – Write-through: processor writes and cache updates itself and mem. Write-back: only update cache, update main mem when block is removed from cache.

• The Carry flag (CF) is set when the result of an unsigned arithmetic operation is too large to fit into the destination.

• The Overflow flag (OF) is set when the result of a signed arithmetic operation is too large or too small to fit into the destination.

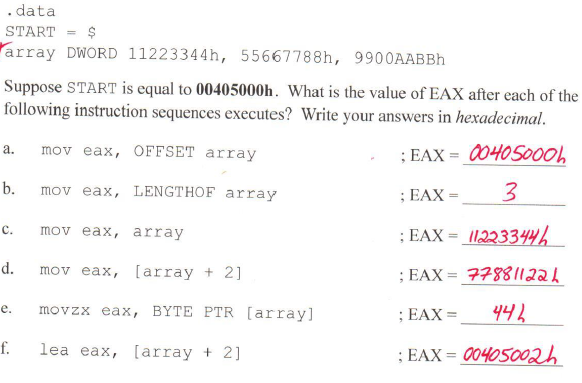
• The Sign flag (SF) is set when the result of an arithmetic or logical operation generates a negative result.

• The Zero flag (ZF) is set when the result of an arithmetic or logical operation generates a result of zero.

• The Auxiliary Carry flag (AC) is set when an arithmetic operation causes a carry from bit 3 to bit 4 in an 8-bit operand.

• The Parity flag (PF) is set if the least-significant byte in the result contains an even number of 1 bits. Otherwise, PF is clear. In general, it is used for error checking when there is a possibility that data might be altered or corrupted.

Sub & cmp flags –



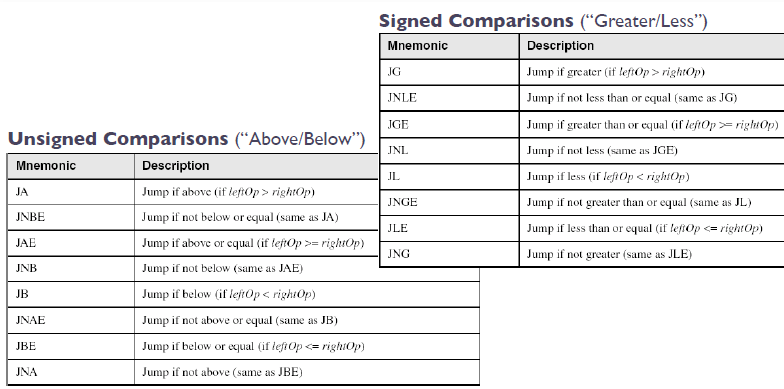
Dest < Src, ZF = 0, CF = 1.

Dest = Src, ZF = 1, CF = 0.

Dest > Src, ZF = 0, CF = 0.

Stack Frame –

Push ebp



Mov ebp, esp

OR enter 0, 0

To end:

Pop ebp or leave

