Powers of 2:

- $1, 2, 4, 8, 16, 32(5), 64, 128, 256, 512, 1024(10)$
- Prefixes (by powers of 2^10): kibi(10), mebi(20), gibi(30), tebi(40), pebi(50), exbi(60), zebi(70), yobi(80)
- Split power of two into tens and ones, use first line to get number, second line to get name. ie, $2^23 = 2^3$ mebi = 8 mb.

MIPS Basics:

- 32 registers, 32 bits for data or instructions
- R instruction: 0 opcode, 2^6 funct codes. 61 regular, 3 for srl, sll, sra
- I instruction: 61 opcodes, mirror of R. branch limits: PC - 2^17 to PC + 2^17 -4.
- J instruction: 2 opcodes (j, jal), target is PC[32:29](target ¡¡ 2). limits: 2^28 (?)
- Declarations. Text = machine code , Data = binary rep of data , word = 32 bit quantities in order.
- Memory - code at bottom, then static, then heap (grows up). Stack at top, grows down.

Conditionals:

- blt (<) = slt, bne t 0 TARGET
- ble (<=) = slt, beq t 0 TARGET
- bgt (>) = slt flip, bne
- bge (>=) = slt flip, beq

Gotchas:

- Remember to sll by 2 when adding counters to pointers - we need word alignment!
- Iterative vs recursive - good compiler can optimize well, so
- Prologue and Epilogue - save ra and all non temp vars to stack, then restore.
- BNE has range 16+2, since last 2 bits are zeros. Also, 2's complement.
-

Sample Code: swap: sll $a1, $a1, 2 # word align i1 sll $a2, $a2, 2 # word align i2 addu $a1, $a0, $a1 # arr+i1 addu $a2, $a0, $a2 # arr+i2 lw $t0, 0($a1) # temp = *(arr+i1) lw $t1, 0($a2) # temp2 = *(arr+i2) sw $t0, 0($a2) # *(arr+i2) = temp sw $t1, 0($a1) # *(arr+i1) = temp2 jr $ra # return

void swap(int * arr, int i1, int i2) int t = arr[i1]; // use t ¡–¿ $t0 arr[i1] = arr[i2]; arr[i2] = t;

entry_label: addi $sp, $sp, -framesize; sw $ra, framesize-4($sp) ... exit_label: lw $ra, framesize-4($sp); addi $sp, $sp, framesize; jr

Bit Twidling:

- AND : 0, 0, 0, 1. used to turn bits OFF (X AND 0)
- OR : 0, 1, 1, 1. Used to turn bits ON (X AND 1)

Average access time formula - get from homework. C - 0 is false. is null.

Caches:

- Compute cache size = $2^(indexbits + rowbits)$
-
- Temporal locality - youngest element likely touched again
- Spatial locality - memory reads are often sequential or at least close in space.
- Dirty bit for write-back, valid bit for tag.
- Direct Mapped - each memory location associated with exactly ONE location - ie, 4 spots and mod 4.
- N-Way Set Associative - map to rows, but rows have N blocks. 2-way gives great perfomance boost, avoid ping pong. Remove oldest elem. (LRU)
- Fully Associative - blocks go anywhere. Hard to find stuff.
- Offset - # bytes is ln width of cache (4 words, 2 bits)
- Index - # bytes is ln # rows in cache (32 rows, 5 bits)
- Tag - what remains. (32 - 2 -5 = 25 bits.)
- Write through - write to cache and memory
- write-back - write to memory on flush, uses dirty bit.
- bits = tag + valid bit + (dirty bit) + data (8*B)

Miss Types:

- Compulsory - cache empty, so everything misses
- Conflict - two blocks map to same space, so forced to swap out
- Capacity - cache is small, so we often run out fo space.

Running a Program (CALL):

- Compile - High level to low level. C → MIPS.
- Assemble - Outputs Object code and infomration tables. Replaces pseudo instructions.
- Link - Input: object file header, text, data, reloc info, symbol table, debug info. Output executable code.
- Load - Allocates space, copies code and static vars into mem, init regs and set stack pointer.

Floating point:

- sign bit, exponent bits, then significand.
-

1. Relax. This test will not matter in 5 years.
2. You survived 61a and 61b, you can do this too.
3. The "A" is yours.

CS61c Midterm 1 Study Guide, by Ivan "Vania" Smirnov
BTC donation address: 14i5jja4KLoUD9jTetbz4AEtTV9Ud1WpsW