CS241

ANDREW CODISPOTI

1. Binary and Hexadecimal numbers

(1) bit – binary digits 1 and 0 (all computer understands)
(2) byte – 8 bits
(3) word
    (a) machine specific grouping of bits
    (b) assume 32-bit architecture
    (c) 1 word = 32 bits= 4 bytes
(4) nibble – 4 bits half a byte

1.1. **Given a byte(or word) in memory what does it mean?** Could mean many things.

(1) A number (which number?)

1.2. **How can we represent negative numbers?** Simply use a sign bit with 0 for + and 1 for - (Sign-Magnitude representation) but then you have two -1's and arithmetic is tricky

1.2.1. *Two's Complement notation.* Interpret the n-bit number as a an unsigned int. If first bit is 0 done else subtract 2^n

n bits- represent $-2^{n-1} \ldots 2^{n-1}$ with left bit still giving sign. arithmetic is clean, just mod 2^n

We cant tell if a number is signed unsigned or two's complement and we have to remember.

We don't even know if what it means:a number, a character, An instruction (or part of one), Garbage

1.3. **Hexadecimal notation.**

(1) base 16 0-9, A-F
(2) more compact than binary
(3) each hex digit = 4 bits (1 nibble)
(4) e.g. 1100 1001 = C9
(5) NOTATION: 0xC9

1.4. **Mapping from binary to characters.**

1.4.1. *ASCII.* Uses 7 bits

IBM implemented extended ascii to use all 8-bits, but they add some weird characters i.e. frame like characters.Compatibility issues because no one standard.

11001001 is not 7 bit ascii, 01001001 decimal 73 is ASCII for I

other standards like EBCDIC

2. Machine Language

Computer programs operate on data and are data(occupy same space as data)

2.1. **Von Neumann architecture.** Programs reside in the same memory as data. Programs can operate on other programs i.e OS

2.2. **Central Processing Unit.** see physical notes for diagram

(1) Control Unit
   (a) decodes instructions
   (b) dispatches to other parts of the computer to carry out instructions
(2) Arithmetic logic unit: Does Math

2.3. **Memory–Many Kinds (Ranked in speed order).**

(1) **CPU
(2) cache
(3) **main memory RAM
(4) disk memory
(5) network memory

2.4. **Registers.** On the CPU, small amount of very fast memory called registers

MIPS 32 General purpose registers $0 to $31

(1) each holds 32 bits
(2) can only operate on data that is in regs.
(3) $0 is always 0
(4) $31 is special and $30
(5) EX: add the contents of two registers and put the result in another register.
(6) 5 bits encode a register $2^5 = 32$
(7) 15 bits to encode registers, 17 bits to encode operation

2.5. **RAM.**

(1) large amount of memory away from cpu
(2) travels between the cpu and ram on the bus
(3) big array of n-bytes, n 10^9
(4) each cell has an address 0,. . . , n-1
(5) each 4-byte block of the form is a word(see diagram 2 in notes)
(6) word addresses are 0,4,8,c,10,14,18,1c
(7) RAM access much slower than reg access

2.6. **Communicating with RAM.** two commands

(1) load
   (a) transfer a word from an address to a register. desired address goes into the memory address register(MAR), goes out on bus
   (b) data at that location comes back on the bus, goes into the memory data register (MDR)
   (c) value in MDR moved to destination register
(2) store: does the reverse of load

2.7. **How does a computer know which words contain instructions and which contain data?** It Doesn't

2.8. **How does it run.** Special register called pc(program counter) which stores the address of the next instruction to execute instruction to execute.

By convention, guarantee that some address(i.e. 0) contains code, initialize pc to 0.

Computer then runs the fetch-execute cycle

```
PC <- 0
loop
  IR <- MEM[PC]
  PC <- 4
  decode and execute the instruction in IR
end loop
```