EECS 370 DISCUSSION #1

Two's Complement, Basic C, LC-2K ISA

(v3)

TWO'S COMPLEMENT

Two's Complement

- Method of representing negative number using binary bits.
- MSB = sign bit.
  - 0 = positive
  - 1 = negative

Numbers “wrap around” to negatives

0111 1110 = 126

Numbers “wrap around” to negatives

0111 1111 = 127

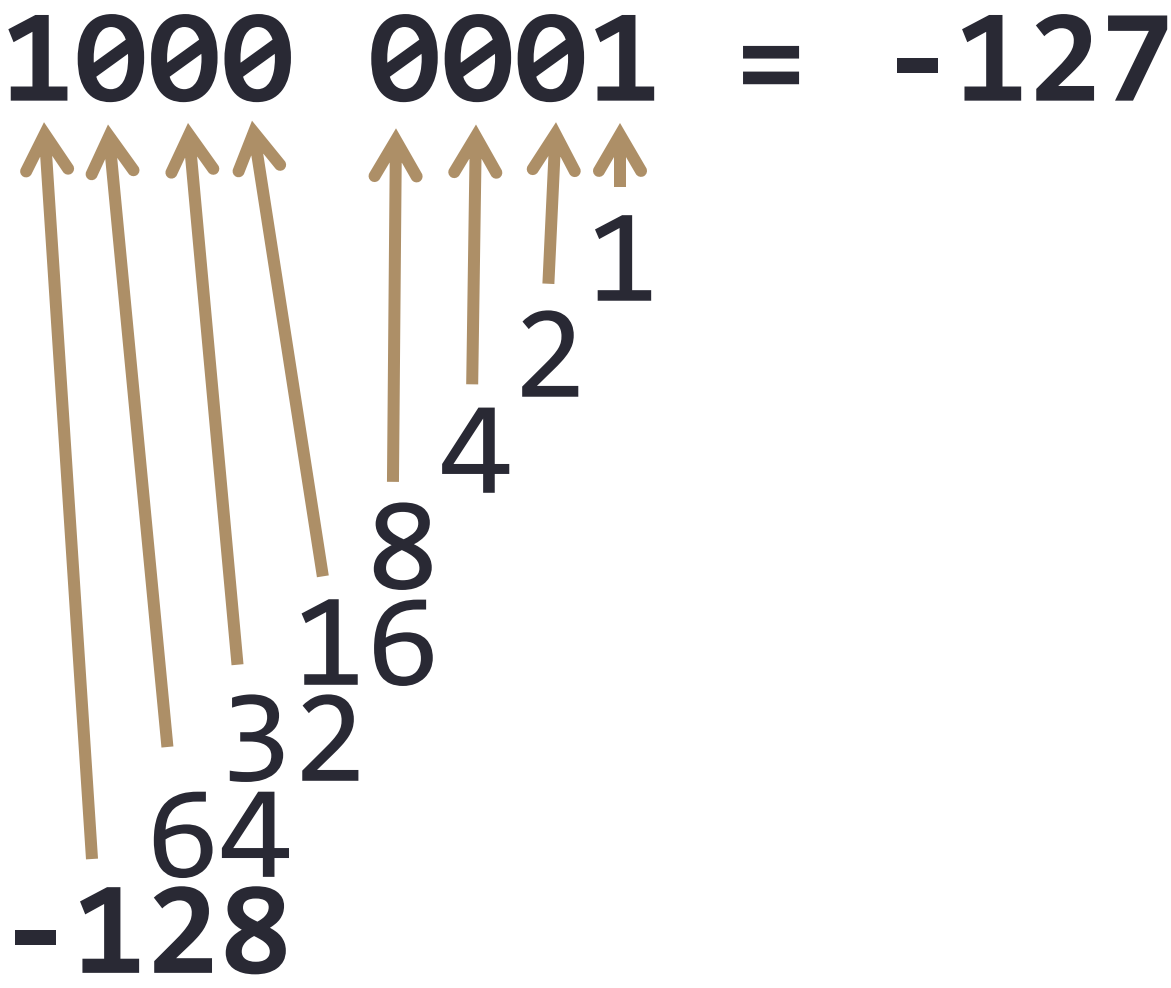
Numbers “wrap around” to negatives

1000 0000 = -128

Numbers “wrap around” to negatives

1000 0001 = -127

Another way to think about it



Negating a Number

1. Original Number

```
0101 1110 = 94
```

2. Invert the bits

```
1010 0001 = -95
```

3. Add 1

```
1010 0010 = -94
```

More examples

- What are these numbers in decimal? (as signed char)
  - `1010 1100` `-84`
  - `1000 0111` `-121`
  - `1100 1010` `-54`

- Negate these numbers:
  - `1001 0111` `0110 1001`
  - `0011 1011` `1100 0101`
  - `0100 0100` `1011 1100`

Converting between Hex and Binary

1. Original Number

```
10101011011100010101011
```

2. Split into 4-bit sections (“Nibbles”)

```
1010 1011 0111 0000 1010 1011
```

3. Convert each Nibble to hex digit (0 – F)

```
0xAB70AB
```

ABOUT C

Language features

- No class objects – only structs.
  - No member functions
  - No public/private

C++	C

class Student
{
private:
    char *name;
    double gpa;
    int classesTaking;
};
	struct Student
	{
	    char *name;
	    double gpa;
	    int classesTaking;
	};

Language features

- No new/delete
  - void *malloc(int) – Creates memory buffer
  - void free(void *) – Deallocates memory buffer
  - **Freeing memory is not a requirement for 370 projects.**
    - But try to do it anyway.

C++	C

`char *buf = new char[100];`
 	
`readIntoBuffer(buf);`
`doStuffWithString(buf);`
 	
`delete [] buf;`
	`char *buf = malloc(100);`
	
	`readIntoBuffer(buf);`
	`doStuffWithString(buf);`
	
	`free(buf);`

Language features

- No declarations in for loops.
  - Declare variables before the for loop.

The following C++ code will not compile in C:

C++	C

`for (int x = 0; x < 25; ++x)`
`  singSongAtIndex(x);`
	`int x;`
	`for (x = 0; x < 25; ++x)`
	`  singSongAtIndex(x);`

- Alternatively, compile in gcc with –std=c99

Language features

- No bool type
  - Use int instead*

C++	C

`bool valid = true;`
`if (a)`
`  valid = false;`
`if (valid && !b)`
`  valid = false;`
 	
`if (valid)`
`  doIt();`
	`typedef int bool;`
	`const bool true = 1;`
	`const bool false = 0;`
	
	`/* Same code here */`

- Alternatively, #include <stdbool.h> (except Windows)

String Handling

- C has no string class.
- You must use char arrays. (char *)

C++	C
`string name;` `name = “Walter White”;`	`const char *name;` `name = “Walter White”;`

String Functions

- strlen(), strcmp(), strcpy()

C++	C

`string s1 = getFirstString();`
`string s2 = getSecondString();`
`string s3 = getThirdString();`
 	
`// C++ string class handles`
`// buffer allocation for you.`
`s2 = s3;`
 	
`int size = s1.length();`
 	
`if (s1 == s3)`
`  theyAreTheSame();`
	`char *s1 = getFirstString();`
	`char *s2 = getSecondString();`
	`char *s3 = getThirdString();`
	
	`// s2 must be large enough`
	`// to hold the contents of s3.`
	`strcpy(s2, s3);`
	
	`int size = strlen(s1);`
	
	`if (strcmp(s1, s3) == 0)`
	`  theyAreTheSame();`

Additional String Functions

- char *strdup(const char *something)
  - Include <string.h> (same for strlen, strcmp, …)
  - Makes a copy of a string in memory.

C++

```
char *orig = “Recon.\n”;
char *extra = strdup(orig);

extra[0] = ‘B’;
extra[1] = ‘a’;

cout << orig << extra;

// Output:
//      Recon.
//      Bacon.
```

I/O in C – Output

- Output is produced through printf() function

C	Output

`int age = 19;`
`const char *name = “Aaron”;`
`double gpa = 3.81;`
 	
`printf(“Simple output\n”);`
`printf(“Integers: %d”, 45);`
`printf(“Doubles: %f”, 12.45);`
`printf(“Chars: %c”, ‘h’);`
`printf(“Strings: %s”, “welp”);`
 	
`printf(“%s is %d and has a ”`
`“%f gpa”, name, age,`
`gpa);`
	`Simple output`
	`Integers: 45`
	`Doubles: 12.450000`
	`Chars: h`
	`Strings: welp`
	
	`Aaron is 19 and has a 3.810000 gpa.`

I/O in C – Input

- Input can be obtained through scanf() function.
- Uses similar format to printf.
- Pass pointers to where you want data to go.

- Example: reading lines from a file:
  - {UMID} {uniqname} {GPA}

C

```
unsigned int id;
char *uname = malloc(10);
float gpa;

scanf(“%d %s %f”, &id, uname, &gpa);
```

I/O in C – Useful functions

- atoi/atol/atof – Converts char* to int/long/double.
- strtol/strtod – Similar to atol/f, but with more options.
  - You can choose a numeric base. (hex, dec, octal, etc…)
  - Tells you where the conversion stopped.
    (useful for reading multiple things from one line)
- strtok – Breaks a string into multiple tokens
  - Very helpful for assembler.

- Include stdio.h, stdlib.h, string.h for these (and others)

Other I/O Functions

- fprintf/fscanf
  - Same as printf/scanf, but uses a FILE* object to access a file.
- sprintf/sscanf
  - Same as printf/scanf, but uses a char buffer to write/read.
- getline()
  - Reads an entire line of input from a FILE* object.
- Use stdin and stdout as FILE* objects for console I/O.

LC-2K INSTRUCTIONS

LC-2K Instructions - add

- add regA regB destReg

- Effect: [destReg] = [regA] + [regB]
- Example: add 1 2 3 // r3 = r2 + r1

- regA, regB, and destReg MUST be registers (no labels)

LC-2K Instructions - nand

- nand regA regB destReg

- Effect: [destReg] = [regA] NAND [regB]
- Example: nand 1 2 3 // r3 = r2 NAND r1

- regA, regB, and destReg MUST be registers (no labels)
- Implementing in C:
  - a & b means bitwise AND of a and b
  - ~c means bitwise complement (inversion) of c
  - ∴ ~(x & y) means bitwise NAND of x and y

LC-2K Instructions - lw

- lw regA regB offset

- Effect: [regB] = [[regA] + offset]
- Example:
```
lw    4  5  1005
lw    6  1  array
```

- regA, regB MUST be registers (no labels)
- When assembling:
  - offset is number: use numeric value in outputted instruction
  - offset is label: use the address of the label in outputted instruction

LC-2K Instructions - sw

- sw regA regB offset

- Effect: [[regA] + offset] = [regB]
- Example:
```
sw   2  7  8511
sw   5  6  table
```

- regA, regB MUST be registers (no labels)
- When assembling:
  - offset is number: use numeric value in outputted instruction
  - offset is label: use the address of the label in outputted instruction

LC-2K Instructions – noop

- noop = No Operation
- Takes no extra parameters, does nothing.
- Wastes a processor cycle

- C implementation: ;

LC-2K Instructions – halt

- halt

- Effects: Stops the execution of the program.

LC-2K Instructions – jalr

- jalr regA regB

- Effect: Stores (PC + 1) in regB, sets PC to regA.
- Example: jalr 3 4

- regA, regB MUST be registers (no labels)

- Used in function calls
  - You can **lw** the address of a label into some register.
  - Store your current PC value into a register so you can **return**.

LC-2K Instructions – beq

- beq regA regB offset

- Effect: If [regA] == [regB], then PC = PC + 1 + offset
- Example: beq 3 4 Loop

- regA, regB MUST be registers (no labels)

- When assembling:
  - If offset is label, calculate (&label – &beq – 1) as offset.

Techniques - beq

- beq x y 0
  - No-op
- beq x x y
  - Unconditional branch (useful for looping)
- beq x y 1
  - Skips next instruction after BEQ statement.
- beq x x -1
  - Infinite loop!

Techniques - nand

- Let a N b = a NAND b

- Remember logic laws from EECS 203
- a N a = NOT a (bitwise complement)
- (a N b) N (a N b) = a AND b
- (((aNa)N(bNb))N((aNa)N(bNb)))N(((aNa)N(bNb))N((aNa)N(bNb))) = a OR b ！

- How do you negate a number in LC-2K?

Techniques – lw/sw

Given some program:

```
        add ...
        nand ...
array   .fill 1
        .fill 3
        .fill 9
        .fill 27
        .fill 81
        .fill 243
        .fill 729
        ...
```

- Suppose r7 contains some value of interest.
- Use lw/sw as arrays / lookup tables.
- Compute r4 = 3^{r7}:
  - lw 7 3 array
    - e.g. r3 = array[r7]

Example – Converting C to LC-2K

C	LC-2K

```
int a = 5;
int b = 10;

while (a != b)
{
        a ++;
}
```

```
        lw      0       1       five
        lw      0       2       ten
        lw      0       3       one
loop    beq     1       2       exit
        add     1       3       1
        beq     0       0       loop
exit    halt
one     .fill   1
ten     .fill   10
five    .fill   5
```

- Start by choosing registers to use for variables.
- Remember beq target = PC (of beq stmt) + 1 + offset
  - 1st beq is transformed to “beq 1 2 2”
  - 2nd beq is transformed to “beq 0 0 -3”