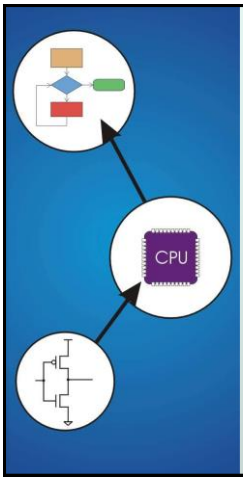

Midterm Exam
Review Slides

Original slides from Gregory Byrd, North Carolina State University

Modified slides by Chris Wilcox, Colorado State University

Review Topics

- Number Representation
- Base Conversion
- Floating-Point Math
- 2's Complement Arithmetic
- Bitwise Operators
- C Programming

Number Representation
What can a binary number mean?

- Interpretations of a 32-bit memory location:
  - 32-bit floating point (IEEE)
  - 32-bit unsigned/signed integer
  - 16-bit unsigned/signed integer (2)
  - 8-bit unsigned/signed bytes (4)
  - ASCII characters (4)
  - RISC instruction
  - Control or status register
  - .jpg. .mpg, .mp3., .avi, …

Number Representation
Hexadecimal to Binary Conversion

- Method: Convert hexadecimal digits to binary using table.
- Question: What is hexadecimal 0xFEBD4570 in binary?

```
F    E    B    D    4    5    7    0
1111 1110 1011 1101 0100 0101 0111 0000
```

- Answer:

```
11111110101111010100010101110000
```

Hexadecimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Number Representation
Binary to Hexadecimal Conversion

- Method: Group binary digits, convert to hex digits using table.
- Question: What is binary 1100110111101111000100100011000 in hexadecimal?

```
1100 1101 1110 1111 0001 0010 0011 0000
C    D    E    F    1    2    3    0
```

- Answer: `0xCDEF1230`

Hexadecimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Number Representation
Decimal to Binary Conversion

- Method: Convert decimal to binary with divide by 2, check odd/even.
- Question: What is decimal 49 in binary?

```
49 is odd, prepend a '1'                1
49 / 2 = 24 is even, prepend a '0'     01
24 / 2 = 12 is even, prepend a '0'    001
12 / 2 = 6 is even, prepend a '0'    0001
6 / 2 = 3 is odd, prepend a '1'     10001
3 / 2 = 1 is odd, prepend a '1'    110001
```

Answer: `110001`

2^n	Decimal
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024

Number Representation
Binary to Decimal Conversion

- Method: Convert binary to decimal by multiplying by 2, add 1 if bit set.
- Question: What is binary 110101 in decimal?

```
Start with 0                            0
Left bit set, multiply by 2, add 1      1
Left bit set, multiply by 2, add 1      3
Left bit clear, multiply by 2           6
Left bit set, multiply by 2, add 1     13
Left bit clear, multiply by 2          26
Left bit set, multiply by 2, add 1     53
```

Answer: `53`

2^n	Decimal
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024

Number Representation
Binary to Floating Point Conversion

- Single-precision IEEE floating point number:

`1 01111111 10000000000000000000000`

sign *exponent* *fraction*

  - Sign is 1 – number is negative.
  - Exponent field is 01111111 = 127 – 127 = 0 (decimal).
  - Fraction is 1.100000000000… = 1.5 (decimal).

- Value = $-1.5 \times 2^{(127-127)} = -1.5 \times 2^0 =$ **-1.5**

Number Representation
Floating Point to Binary Conversion

- Value = 6.125
  - Number is positive – sign is 0
  - Fraction is 110.001 (binary), normalize to 1.10001 * 2^2
  - Exponent is 2 + 127 = 129 (decimal) = 10000001
- Single-precision IEEE floating point number:

0 10000001 10001000000000000000000

sign exponent fraction

Number Representation
Hexadecimal to ASCII Conversion

- Method: Convert values to ASCII by table lookup.
- Each two (hex) digits is a single character.
- Question: What is hex 0x42454144 in ASCII?

```
0x42 = 'B'
0x45 = 'E'
0x41 = 'A'
0x44 = 'D'
```

- Answer: "BEAD"

Char	ASCII Code	Char	ASCII Code
'A'	0x41	'0'	0x30
'B'	0x42	'1'	0x31
'C'	0x43	'2'	0x32
'D'	0x44	'3'	0x33
'E'	0x45	'4'	0x34
'F'	0x46	'5'	0x35
'G'	0x47	'6'	0x36

Computer Arithmetic
Signed Integer Representations

Binary Number	Signed Magnitude	1's Complement	2's Complement
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

Computer Arithmetic
2's Complement Arithmetic

- Binary Arithmetic (unsigned integers):

```
  1 0 0 1 0 0 1 0
+ 0 0 1 1 0 1 0 1
-----------------
0 1 1 0 0 0 1 1 1
```

- Hex Equivalent:

0x92 + 0x35 = 0xC7

- Decimal Equivalent:

146 + 53 = 199

- Binary Arithmetic (signed integers):

```
  1 0 0 1 0 0 1 0
+ 0 0 1 1 0 1 0 1
-----------------
0 1 1 0 0 0 1 1 1
```

- Hex Equivalent:

0x92 + 0x35 = 0xC7

- Decimal Equivalent:

-110 + 53 = -57

Computer Arithmetic
Bitwise Logical Operations

- Bitwise AND (&):

```
  1 1 1 1 0 0 0 0
& 0 0 1 1 0 1 0 1
  0 0 1 1 0 0 0 0
```

- Hex Equivalent:

0xF0 & 035 = 0xC0

- Bitwise OR (|):

```
  1 1 1 1 0 0 0 0
| 0 0 1 1 0 1 0 1
  1 1 1 1 0 1 0 1
```

- Hex Equivalent:

0xF0 | 035 = 0xF5

C Programming
Bit Manipulation

- C code to read or write a bit:

```
int readBit(int value, int bit) {
      return (value >> bit) & 01;
      // return !!(value >> bit);
}

void writeBit(int *value, int bit) {
      *value |= 1<<bit;
}
```


C Programming
Control Structures

- C conditional and iterative statements
  - if statement

```
if (value == 0x12345678)
    printf("value matches 0x12345678\n");
```

  - for loop

```
for (int i = 0; i < 8; ++i)
    printf("i = %d\n", i);
```

  - while loop

```
int j = 6;
while (j--)
    printf("j = %d\n", j);
```


C Programming
Basic Pointers

- C pointers

```
void foo(int *intp, double *doublep){
    *intp = 28;
    *doublep = 2.34;
}
int main(int argc, char *argv[]) {
   int i = 17;
   double d = 1.23;
   foo(&i, &d);
   printf("%i, %.2f\n",i, d);
}
// prints 28,2.34
```

C Programming
Pointers and Arrays

- C pointers and arrays

```
void foo(int *pointer)
{
    *(pointer+0) = pointer[2] = 0x1234;
    *(pointer+1) = pointer[3] = 0x5678;
}
int main(int argc, char *argv[])
{
   int array[]= {0, 1, 2, 3};
   foo(array);
   for (int i = 0; i <= 3; ++i)
      printf("array[%d] = %x\n", i, array[i]);
}
```


C Programming
Memory Allocation

- Static Allocation

```
char cArray[100];
char cArray[] = {'a', '%', '5'};
```

- Dynamic Allocation

```
char *cArray = malloc(100 * sizeof(char));
char *cArray = calloc(100, sizeof(char));
cArray[index] = 1234;
free (cArray);
```