

Final Exam Review Slides

Original slides from Gregory Byrd, North Carolina State University
Modified slides by Chris Wilcox, Colorado State University

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Review Topics

- Number Representation
- Transistors and Gates
- Combinational Logic
- LC-3 Programming
- Memory Model
- C Programming

CS270 - Fall Semester 2015 2

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Number Representation Hexadecimal to Binary Conversion

- Method: Convert hexadecimal digits to binary using table.
- Question: What is hexadecimal **0x1234CDEF** in binary?

1 2 3 4 c d e f
0001 0010 0011 0100 1100 1101 1110 1111

- Answer:
00010010001101001100110111101111

| 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 |

CS270 - Fall Semester 2015

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Number Representation Binary to Hexadecimal Conversion

- Method: Group binary digits, convert to hex digits using table.
- Question: What is binary **01100111100010011111111011011010** in hexadecimal?

0110 0111 1000 1001 1111 1110 1101 1010
6 7 8 9 F E D A

- Answer: 0x6789FEDA

| 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 |

CS270 - Fall Semester 2015

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?

$$2^5 + 2^4 + 2^0 = 32 + 16 + 1 = 49$$

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 Two's Complement

- Invert all the bits, and add 1.
- Question: What is the value -8 in (16-bit) 2's complement form?

$$8 = 0x0008 = 0000\ 0000\ 0000\ 1000$$

$$\text{Invert bits} = 1111\ 1111\ 1111\ 0111$$

$$\text{Add one} = + 0000\ 0000\ 0000\ 0001$$

$$\text{Answer} = 11111111111111000 \text{ (binary)}$$

$$\text{Answer} = 0xFFFF8$$

Answer: **0xFFFF8 = -8 decimal**

Number Representation Binary to Floating Point Conversion

- Single-precision IEEE floating point number:

1 01111110 1000000000000000000000

↑ ↑ ↑
sign exponent fraction

- Or 0xBF400000
- Sign is 1 – number is negative.
- Exponent field is 01111110 = 126 – 127 = -1 (decimal).
- Fraction is 1.100000000000... = 1.5 (decimal).

- Value = $-1.5 \times 2^{(126-127)} = -1.5 \times 2^{-1} = -0.75$

Number Representation Floating Point to Binary Conversion

- Value = **4.25**
 - Number is positive – sign is 0
 - Fraction is 100.01 (binary), normalize to 1.0001×2^2
 - Exponent is $2 + 127 = 129$ (decimal) = **1000001**

- Single-precision IEEE floating point number:

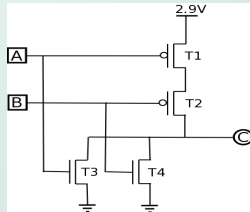
0 1000001 0001000000000000000000

↑ ↑ ↑
sign exponent fraction

- or **0x40880000**

Transistors and Gates

NOR Gate



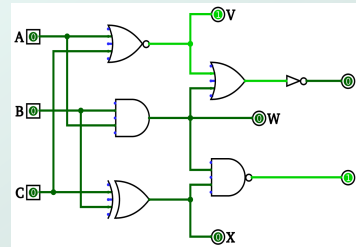
| A | B | T1 | T2 | T3 | T4 | C |
|---|---|--------|--------|--------|--------|---|
| 0 | 0 | Closed | Closed | Open | Open | 1 |
| 0 | 1 | Closed | Open | Open | Closed | 0 |
| 1 | 0 | Open | Closed | Closed | Open | 0 |
| 1 | 1 | Open | Open | Closed | Closed | 0 |

CS270 - Fall Semester 2015

9

Combinational Logic

Combinational Circuit to Truth Table



| A | B | C | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |

CS270 - Fall Semester 2015

10

LC-3 Architecture

Instruction Set (First Half)

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------------------|------|-----|------------|--------|-----------|-----|---|---|---|---|---|---|---|---|---|---|
| ADD ⁺ | 0001 | DR | SR1 | 0 | 00 | SR2 | | | | | | | | | | |
| ADD ⁺ | 0001 | DR | SR1 | 1 | imm5 | | | | | | | | | | | |
| AND ⁺ | 0101 | DR | SR1 | 0 | 00 | SR2 | | | | | | | | | | |
| AND ⁺ | 0101 | DR | SR1 | 1 | imm5 | | | | | | | | | | | |
| BR | 0000 | n | z | p | PCoffset9 | | | | | | | | | | | |
| JMP | 1100 | 000 | BaseR | 000000 | | | | | | | | | | | | |
| JSR | 0100 | 1 | PCoffset11 | | | | | | | | | | | | | |
| JSRR | 0100 | 0 | 00 | BaseR | 000000 | | | | | | | | | | | |
| LD ⁺ | 0010 | DR | PCoffset9 | | | | | | | | | | | | | |

CS270 - Fall Semester 2015

11

LC-3 Architecture

Instruction Set (Second Half)

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------------------|------|------------|-----------|---------|----|----|---|---|---|---|---|---|---|---|---|---|
| LDI ⁺ | 1010 | DR | PCoffset9 | | | | | | | | | | | | | |
| LDR ⁺ | 0110 | DR | BaseR | offset6 | | | | | | | | | | | | |
| LEA ⁺ | 1110 | DR | PCoffset9 | | | | | | | | | | | | | |
| NOT ⁺ | 1001 | DR | SR | 111111 | | | | | | | | | | | | |
| RET | 1100 | 000 | 111 | 000000 | | | | | | | | | | | | |
| RTI | 1000 | 0000000000 | | | | | | | | | | | | | | |
| ST | 0011 | SR | PCoffset9 | | | | | | | | | | | | | |
| STI | 1011 | SR | PCoffset9 | | | | | | | | | | | | | |
| STR | 0111 | SR | BaseR | offset6 | | | | | | | | | | | | |
| TRAP | 1111 | 0000 | trapvect8 | | | | | | | | | | | | | |

CS270 - Fall Semester 2015

12

LC-3 Architecture Addressing Modes

- ◆ Load -- read data **from memory to register**
 - **LD**: PC-relative mode
 - **LDR**: base+offset mode
 - **LDI**: indirect mode
- ◆ Store -- write data **from register to memory**
 - **ST**: PC-relative mode
 - **STR**: base+offset mode
 - **STI**: indirect mode
- ◆ Load pointer: **compute address, save in register**
 - **LEA**: immediate mode
 - *does not access memory*

LC-3 Architecture Machine Code to Assembly

- What is the assembly code for machine instruction **0101010010111101**?
- Step 1) Identify opcode: **0101** = AND
- Step 2) Parse entire instruction (use reference)
- Step 3) Get values from each field

| OPCODE | DR | SR | 1 | imm5 |
|--------|------|-----|---|-------|
| 15:12 | 11:9 | 8:6 | 5 | 4:0 |
| 0101 | 010 | 010 | 1 | 11101 |
| AND | R2 | R2 | | -3 |

- Step 4) Translate to mnemonics: **AND R2,R2,#-3**

LC-3 Architecture Assembly to Machine Code

- What is the machine code for assembly instruction **NOT R7,R6**?
- Step 1) Identify opcode: NOT = **1001**
- Step 2) Put values into each field:

| NOT | R7 | R6 | |
|--------|------|-----|--------|
| OPCODE | DR | SR | imm5 |
| 15:12 | 11:9 | 8:6 | 5:0 |
| 1001 | 111 | 110 | 111111 |

- Step 3) Build machine instruction: **100111110111111**

LC-3 Architecture Assembly Code Syntax

```

.ORIG x3000
MAIN AND R0,R0,#0 ; Initialize Sum
     JSR COMPUTE ; Call function
     ST R0, SUM ; Store Sum
     HALT ; Program complete
COMPUTE LD R1,OPERAND1 ; Load Operand1
        LD R2,OPERAND2 ; Load Operand2
        ADD R0,R1,R2 ; Compute Sum
        RET ; Function return
;; Input data set
OPERAND1 .FILL x1234 ; Operand1
OPERAND2 .FILL x4321 ; Operand2
SUM .BLKW 1 ; Sum
.END
    
```

Memory Model Push and Pop Stack

- Assume POP and PUSH code as follows:

```
MACRO PUSH (reg)
    ADD R6,R6,#-1 ; Decrement SP
    STR reg,R6,#0 ; Store value
END

MACRO POP (reg)
    LDR reg,R6,#0 ; Load value
    ADD R6,R6,#1 ; Increment SP
END
```

Memory Model Detailed Example

- Main program to illustrate stack convention:

```
.ORIG x3000
MAIN LD R6,STACK ; init stack pointer
     LD R0,OPERAND0 ; load first operand
     PUSH R0 ; PUSH first operand
     LD R1,OPERAND1 ; load second operand
     PUSH R1 ; PUSH second operand
     JSR FUNCTION ; call function
     LDR R0,R6,#0 ; POP return value
     ADD R6,R6,#3 ; unwind stack
     ST R0,RESULT ; store result
     HALT
```

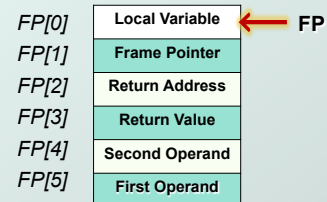
Memory Model Detailed Example

- Function code to illustrate stack convention:

```
FUNCTION
    ADD R6,R6,#-1 ; alloc return value
    PUSH R7 ; PUSH return address
    PUSH R5 ; PUSH frame pointer
    ADD R5,R6,#-1 ; FP = SP-1

    ADD R6,R6,#-1 ; alloc local variable
    LDR R2,R5,#4 ; load first operand
    LDR R3,R5,#5 ; load second operand
    ADD R4,R3,R2 ; add operands
    STR R4,R5,#0 ; store local variable
```

Memory Model Detailed Example



Stack before STR instruction

Memory Model Detailed Example

- Function code to illustrate stack convention:

```

FUNCTION ; stack exit code
    STR R4,R5,#3 ; store return value
    ADD R6,R5,#1 ; SP = FP+1
    POP R5 ; POP frame pointer
    POP R7 ; POP return address
    RET ; return

OPERAND0 .FILL x1234 ; first operand
OPERAND1 .FILL x2345 ; second operand
RESULT .BLKW 1 ; result
STACK .FILL x4000 ; stack address

```

C Programming Bitwise Operators

- C code with bitwise operators:

```

int i = 0x11223344;
int j = 0xFFFF0000;
printf("0x%x\n", ~i); 0xEEDDCBB
printf("0x%x\n", i & j); 0x11220000
printf("0x%x\n", i | j); 0xFFFF3344
printf("0x%x\n", i ^ j); 0xEEDD3344

```

C Programming Logical Operators

- C code with logical operators:

```

int i = 0x11223344;
int j = 0x00000000;
printf("0x%x\n", !i); 0x00000000
printf("0x%x\n", !j); 0x00000001
printf("0x%x\n", i && j); 0x00000000
printf("0x%x\n", i || j); 0x00000001

```

C Programming Arithmetic Operators

- C code with arithmetic operators:

```

int i = 10;
int j = 2;
printf("d\n", i + j); 12
printf("d\n", i - j); 8
printf("d\n", i * j); 20
printf("d\n", i / j); 5

```

C Programming Functions

- C function prototypes
- must precede implementation of function

```
int addInt(int i, int j);
float addFlt(float u, float v);
void addInt(int param0, int param1, int *result);
void addFlt(float f0, float f1, float *result);
bool writeFile(char *filename, Instructions[]);
void input(Instruction *pInstruction);
char *printInt(int number);
```

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-- > 0)
    printf("j = %d\n", j);
```

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 Data Structures

- C data structures

```
// Structure definition
struct sCoordinate
{
    float X;
    float y;
    float z;
}

typedef struct
{
    ...
} Coordinate;
```


C Programming Data Structures

- ◆ C data structures

```
// Structure allocation
struct sCoordinate coordinates[10]; // no typedef
Coordinate coordinates[10]; // typedef
Coordinate *coordinates =
    (Coordinate *) malloc(sizeof(Coordinate) *10);

// Structure access
coordinates[5].X = 1.0f;
pCoordinate->X = 1.0f;
```

C Programming Strings

- ◆ C strings

```
char *string = "hello";
char *carray = { 'h','e','l','l','o' };
char label[20];
strcpy(label, "hello");
strcat(label, " world");
printf("%s\n", label); hello world
printf("%d\n", strlen(label)); 11
printf("%d\n", sizeof(label)); 20
```

C Programming Include Files

- ◆ C include files

```
#include <stdio.h> - FILE, stdout, stdin, stderr,
putchar, getchar, printf, scanf, fprintf, fscanf,
fopen, fclose, ...
#include <stdlib.h> - atof, atoi, malloc, free,
rand, exit, getenv, system, ...
#include <stddef.h> - NULL, size_t, ...
#include <stdbool.h> - bool, true, false
#include <string.h> - memcpy, memset, strcpy,
strcat, strlen, strtok, ...
#include <math.h> - sin, cos, tan, exp, log,
fmod, fabs, floor, ceil, ...
```

C Programming Main Program

- ◆ C include files

- ◆ command line arguments are passed to main
- ◆ arguments are strings, may need to convert
- ◆ getenv function queries environment variables

```
int main(int argc, char *argv[])
{
    printf("%d\n", argc); // # of arguments
    printf("%s\n", argv[0]); // program name
    printf("%s\n", argv[1]); // first argument
    printf("%s\n", argv[2]); // second argument
};
```