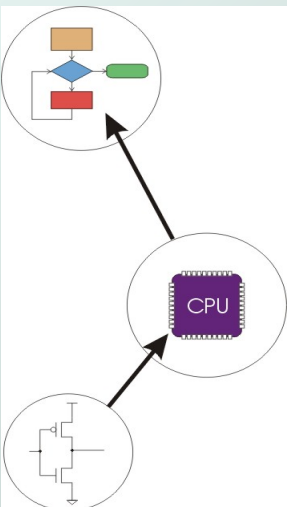


Chapter 3 Digital Logic Structures

Original slides from Gregory Byrd, North Carolina State University
Modified slides by C. Wilcox, S. Rajopadhye Colorado State University

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

Computing Layers



- Problems
-
- Algorithms
-
- Language
-
- Instruction Set Architecture
-
- Microarchitecture
-
- Circuits ←
-
- Devices

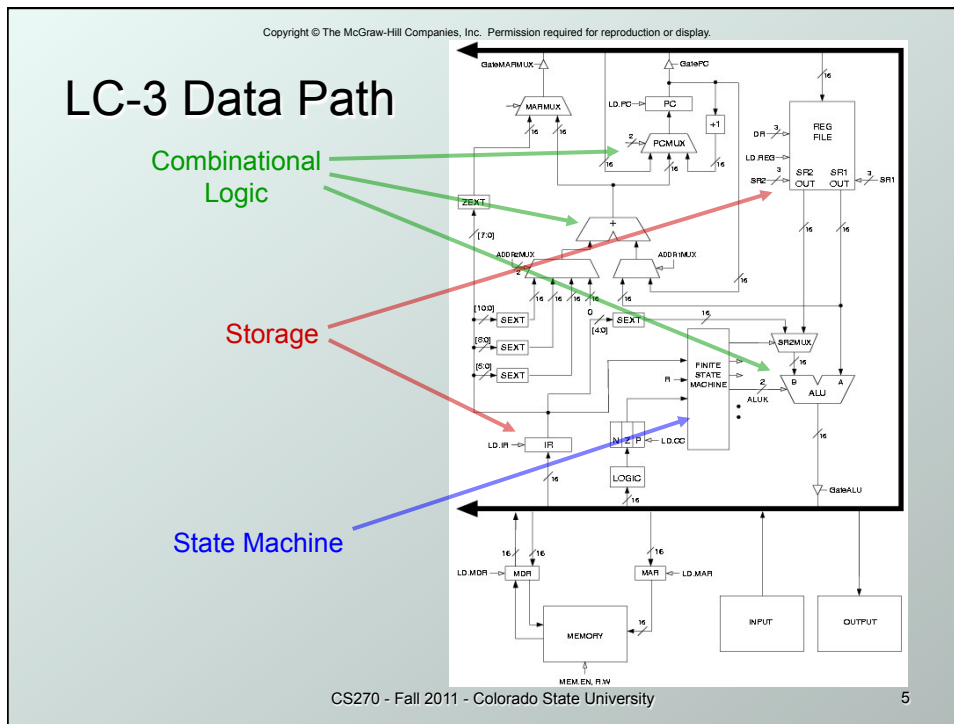
CS270 - Fall 2011 - Colorado State University 2

Fat Combinational Circuits

- **“Fat” Combinational Circuit**
 - A gate with a “bundle of input wires” is actually a “bundle of gates” or a **fat gate**
 - Both bundles must have the same width
 - A single wire on one input is implicitly replicated
 - A combinational circuit built out of fat gates is a fat combinational circuit

From Logic to Data Path

- The data path of a computer is all the logic used to process information.
 - See the data path of the LC-3 on next slide.
- **Combinational Logic**
 - Decoders -- convert instructions into control signals
 - Multiplexers -- select inputs and outputs
 - ALU (Arithmetic and Logic Unit) -- operations on data
- **Sequential Logic**
 - State machine -- coordinate control signals and data movement
 - Registers and latches -- storage elements



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

Looking Ahead: C Functions

- Can pass by value or reference
 - // by value (copies value)**
`float f1(int i, float f);`
 - // by reference (copies pointer)**
`float f2(float *f);`
- Function cannot change values passed by value
 - f1: i = 10; // changes the copy**
- Function can change values passed by reference
 - f2: *f = 1.2; // changes actual value**

CS270 - Fall 2011 - Colorado State University

Looking Ahead: C Arrays

- ◆ Similar to Java arrays

```
// integer array
int iArray[3] = {1,2,3};
printf("iArray[2]: %d", iArray[2]);

// float array
float fArray[2] = {0.1f,0.2f};
printf("fArray[1]: %f", fArray[1]);

// character array
char cArray[4] = {'a','b','c','d'};
printf("cArray[3]: %c", cArray[3]);
```

Looking Ahead: C Strings

- ◆ Array of chars with null termination

```
// string: static allocation
char *string1 = "Hello World\n";
printf("string1: %s", string1);

// string: dynamic allocation
char *string2 = (char *)malloc(13);
strcpy("string2", "Hello World\n");
```

Note that the programmer is responsible for making sure string has enough memory!

Looking Ahead: C Pointers

- Pointers can be used for array access

```
// dynamic allocation for array
int *iArray =
    (int *) malloc(2*sizeof(int));
iArray[0] = 1234; iArray[1] = 5678;
printf("iArray[0]: %d", iArray[0]);
printf("iArray[1]: %d", iArray[1]);
printf("&iArray[0]: %x", &iArray[0]);
printf("&iArray[1]: %x", &iArray[1]);
printf("iArray: %x", iArray);
```

Looking Ahead: C Structures

- Structures

```
struct Student
{
    char firstName[80];
    char lastName[80];
    int testScores[2];
    char letterGrade;
};
struct Student student;
struct Student students[10];
```

Looking Ahead: C Structures

- Structures

```
typedef struct
{
    char firstName[80];
    char lastName[80];
    int testScores[2];
    char letterGrade;
} Student;
Student student;
Student students[10];
```

Looking Ahead: C Structures

- Accessing structures

```
void func(Student student)
{
    strcpy(student.firstName, "John");
    student.letterGrade = 'A';
}

void func(Student *student)
{
    strcpy(student->firstName, "John");
    student->letterGrade = 'A';
}
```

Looking Ahead: Makefiles

- File list and compiler flags

```
C_SRCS = main.c example.c
C_OBJS = main.o example.o
C_HEADERS = example.h
EXE = example

GCC = gcc
GCC_FLAGS = -g -std=c99 -Wall -c
LD_FLAGS = -g -std=c99 -Wall
```

Looking Ahead: Makefiles

- File dependencies

```
# Compile .c source to .o objects
.c.o:
    @echo "Compiling C source files"
    $(GCC) $(GCC_FLAGS) $<
    @echo ""

# Make .c files depend on .h files
${C_OBJS}: ${C_HEADERS}
```

Looking Ahead: Makefiles

- ◆ Build target (default)

```
# Target is the executable
pa3: $(C_OBJS)
    @echo "Linking object modules"
    $(GCC) $(LD_FLAGS) $(C_OBJS) -o $(EXE)
    @echo ""
```

Looking Ahead: Makefiles

- ◆ Miscellaneous targets

```
# Clean up the directory
clean:
    @echo "Cleaning up project directory"
    rm -f *.o *~ $(EXE)

# Package up the directory
package:
    @echo "Packing up project directory"
    tar cvf r4.tar ../R4
```


Anonymous Feedback

- Post test cases for PA1 (done)
- Provide more office hours
- What can I do now to prepare for midterm?
 - Practice, program, read
- Please open PA2
- Recitation R3 is really another homework
 - Student taking 200, 270 & two math classes + working, and already understands concept
 - Each HW is about 3% of your grade – recitations are 0.5%