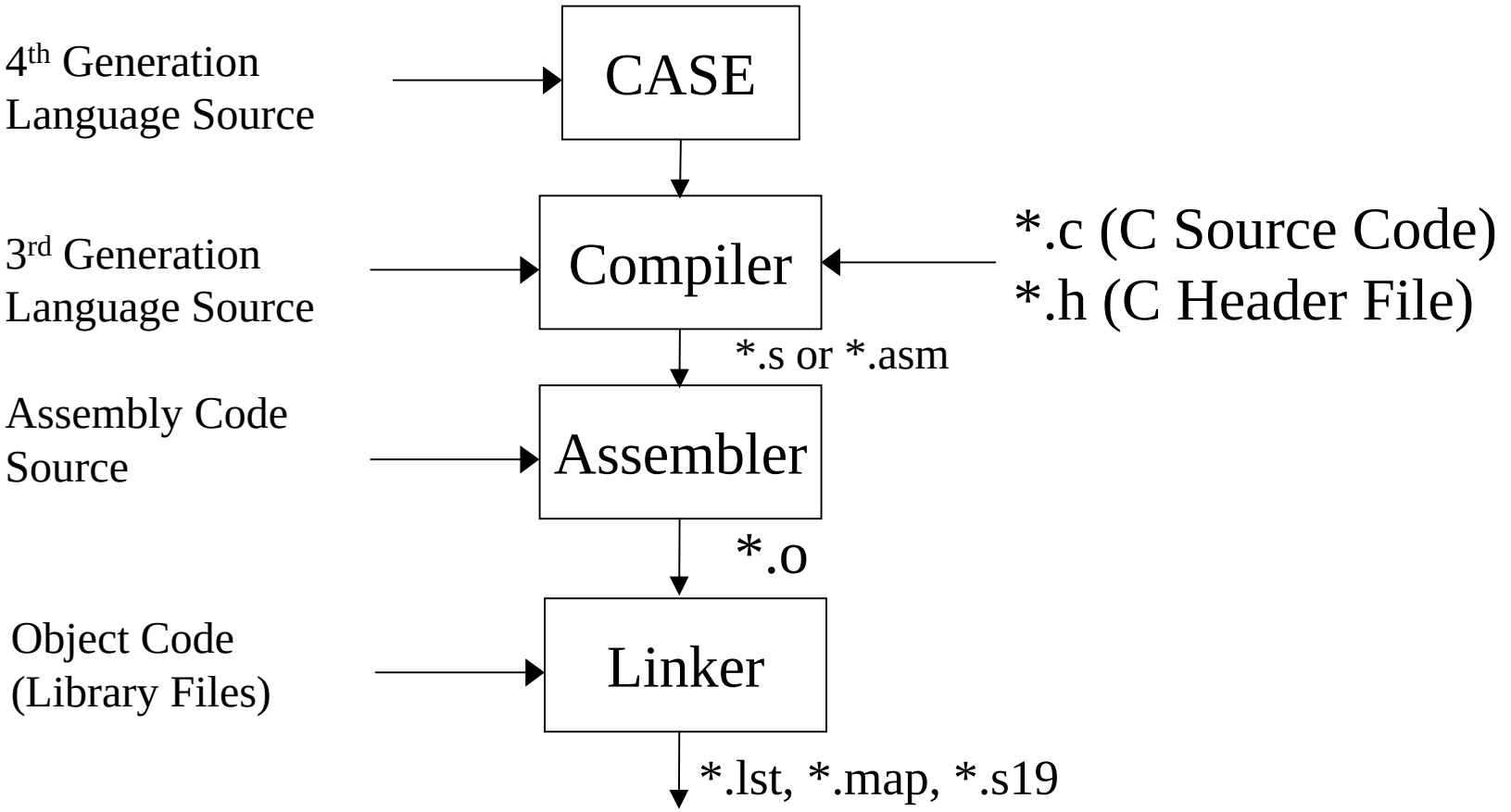
# EE4390 Microprocessors

## Lessons 13-15

## Introduction to C

# Software Programming Tools



# ‘C’ Characteristics

- “Low-level” Language
  - Stays very close to a processor’s hardware.
- Minimalistic
  - Basic single-thread control flow constructs + optional libraries.
- “Lightly-Typed” Language
  - You can do almost anything, including shoot yourself in the foot!
- Function-Based
  - Emphasizes structured design and modularity

# ‘C’ Variable Types

- char
  - Always a single byte. Also used for text.
- int
  - “Usually” matches the size of basic storage unit. Often 2 bytes.
- float
  - Single precision floating point (Used for fractional math)
- double
  - Double precision floating point (Wider range of numbers)

# Variable Modifiers

| | |
|---|---|
| • short | • Remains a 2 byte integer. |
| • long | • Increases number of bytes used for an integer (4 bytes). |
| • signed | • Two's complement numbers (DEFAULT). |
| • unsigned | • Allows unsigned arithmetic. |
| • static | • Directly allocates memory to remember a value between function calls. |
| • extern | • Allows access by other files. |
| • const | • Assigns a constant value to variable. |

# Operators

- Arithmetic

  + - * /

  % modulo (remainder) operator

  EX] 11%3 -> 2

- Relational

  < <= > >=

  == != && ||

  EX] if (num <= 0) || (num > 100)

  {

  :

  }

# Operators

- Bit-wise

  &  |  ^  ~

  EX]  unsigned char a, b;

  a = b & 0x0F;  //0x: hexadecimal

  >>  <<

  EX]  unsigned char a, b;

  a = b >> 2;  //right shift b twice

- Increment/Decrement

  **++  --**

  EX] i++;  equivalent to i = i + 1;

# Bit Twiddling - controlling individual bits

- a | b  bitwise or -  used to turn on specific bits

  EX] PORTA | = 0x80;  //turn on bit 7 (msb)

- a&b bitwise and - useful for checking if specific bits are set

  EX] if ((PORTA & 0x81) == 0) //check bit 7 and 0

- a ^ b bitwise exclusive OR - useful for complementing a bit

  EX] PORTA ^= 0x80;  //flip bit 7

- ~a bitwise complement - used to turn off specific bits

  EX] PORTA &= ~0x80;  //turn off bit 7

# Forcing Variables to Absolute Addresses

```
#pragma abs_address: 0x6000
short m = 0;
int  b = 0;
int temp = 0;
#pragma end_abs_address
```

# while{ } Loops

```
while(expression)
{
   statement1;
   statement2;
}
```

```
k = -10;

while(k < 41)
{
Temp = k * 9/5 + 32;
k++;
printf("Temp: %f \n", Temp);
}
```

# for{ } Loop

```
for (exp1;exp2;exp3)
{
   statement1;
   statement2;
}
```

```
for(k=-10, k<=40; k++)
{
Temp = k * 9/5 + 32;
printf("Temp: %f \n", Temp);
}
```

# do/while {} loop

```
do
{
   statement1;
   statement2;
} while (expression);
```

```
k = -10;
do
{
Temp = k * 9/5 + 32;
k++;
printf("Temp: %f \n", Temp);
} while (k < 41);
```

# if-else-if Statements

```
if(expression)
   statement1;
else if(expression)
   statement2;
else
{
   statement3;
   statement4;
}
```

```
if(c < min_temp)
   flag = TOO_LOW;
else if (c> max_temp)
   flag = TOO_HIGH;
else
{
   temp = v * m + b;
   flag = IN_BOUNDS;
}
```

# Functions

- Function Prototype

  <output type> func_name(<input type1> <variable name1>, ...);

- Function Call

  <variable> = function(<variable name1,...>);

- Function

  output type func_name(<input type1> <variable name1>, ...)

  {

  <Local variables>

  statement1;

  :

  :

  return(output variable);

  }

# Function Example

**Prototype**:

```
int         sqrt(int number);
```

**Call**:

```
sq_root = sqrt(1000);
```

**Function**:

```
int         sqrt(int number)
{
int guess=5, i=1;

for(i=1;guess=5;i != guess;)
   {
    i = guess;
    guess = (i + (10000/i))/2;
    }
return guess;
}
```

# Arrays

- Collection of the same type of data, stored in consecutive memory locations.

- Declaration Examples:

```
int temp[NUM_SAMP];
char msg[] = {“Hello World\n”};
```

- Assignment Example:

```
temp[2] = 50;
```

# Structures

- Collection of multiple variables, possibly of different types.

- Declaration Example:

```
struct circle
{
   int radius;
   int x_center;
   int y_center;
};
```

```
struct circle circle1;
```

# Structures (Cont.)

- Assignment Example:

  circle.radius = 5;

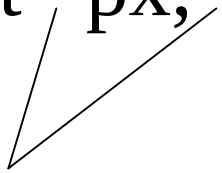
  Employee.name = {“John”};

# Pointers

- Variables assigned to specific addresses.

- Declaring pointers

```
int    x;
int *px, *pa;     //address of integers
```



pointers

# Pointer Syntax

```
px = &x;      //assign pointer to the address of x
```

address of

```
x = *px;      //set x equal to contents of px
```

dereference operator, contents of

# Pointer example

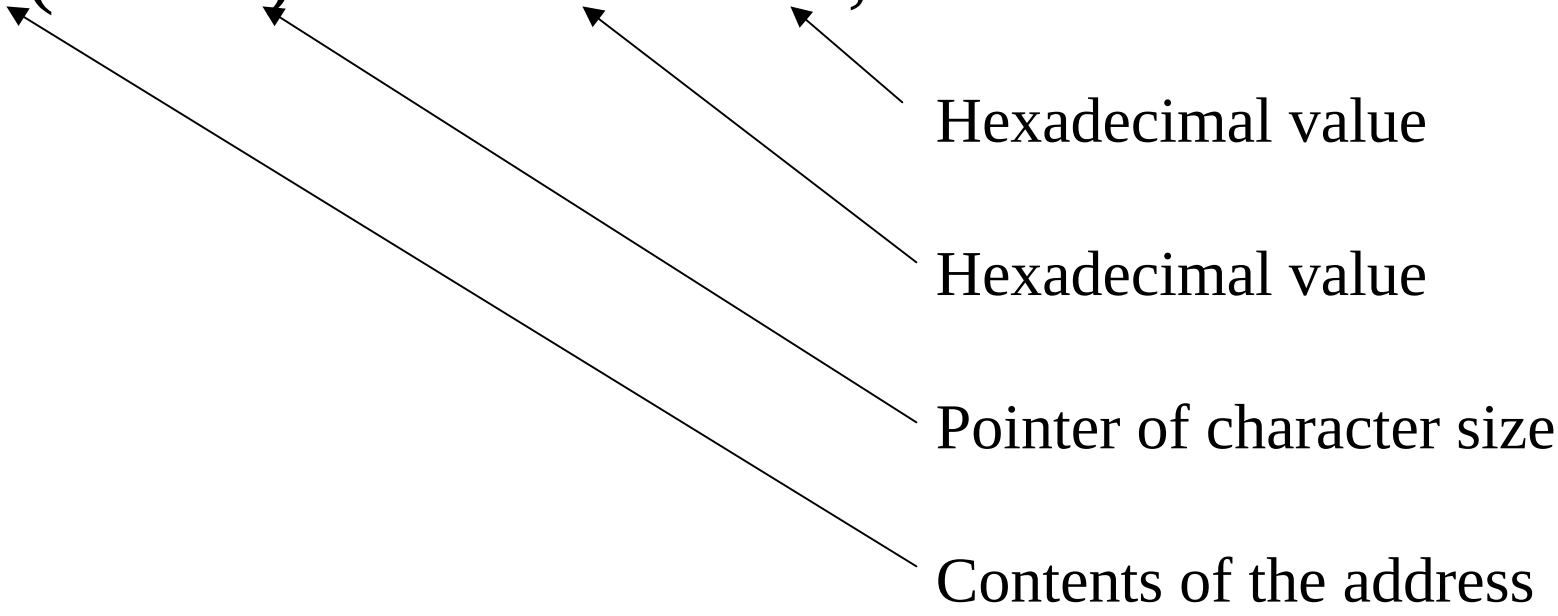
```
int  m,n;      //integers
int  *pm;      //pointer to integer type

m = 10;       //set m equal to 10
pm = &m;  //set integer pointer to addr of m
n = *pm;     //set n equal to contents of pm
                  //n = 10
```

Bottom line: do both sides of equation balance?

# Pointer Syntax

- Placing data at specific locations:

```
* (char *) 0x102B = 0x30;
```



# Pointer Syntax - header files

```
#define _IO_BASE = 0
#define _P(off) *(unsigned char volatile*) (_IO_BASE + off)
#define _LP(off) *(unsigned short volatile*) (_IO_BASE + off)

#define PORTA _P(0x00)
:
:
```

- Allows you to use register file by name in code
- Must include header file with code