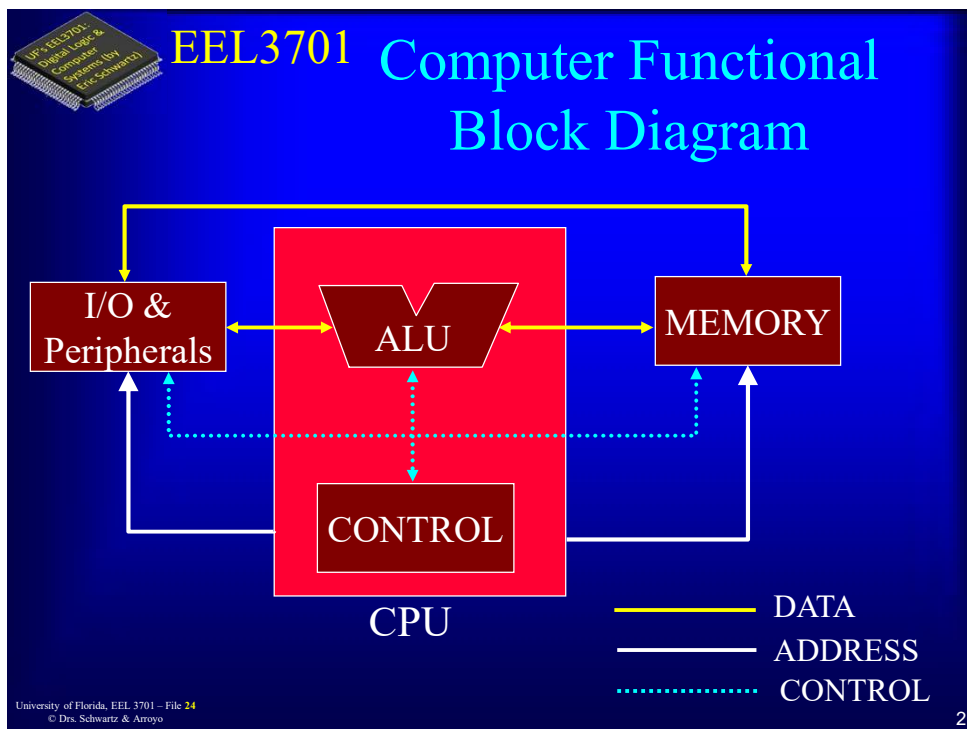# EEL3701

Menu

- Computer Organization
- Programming Model for the an example microprocessors (the G-CPU & Motorola 68HC11)
- Assembly Programming
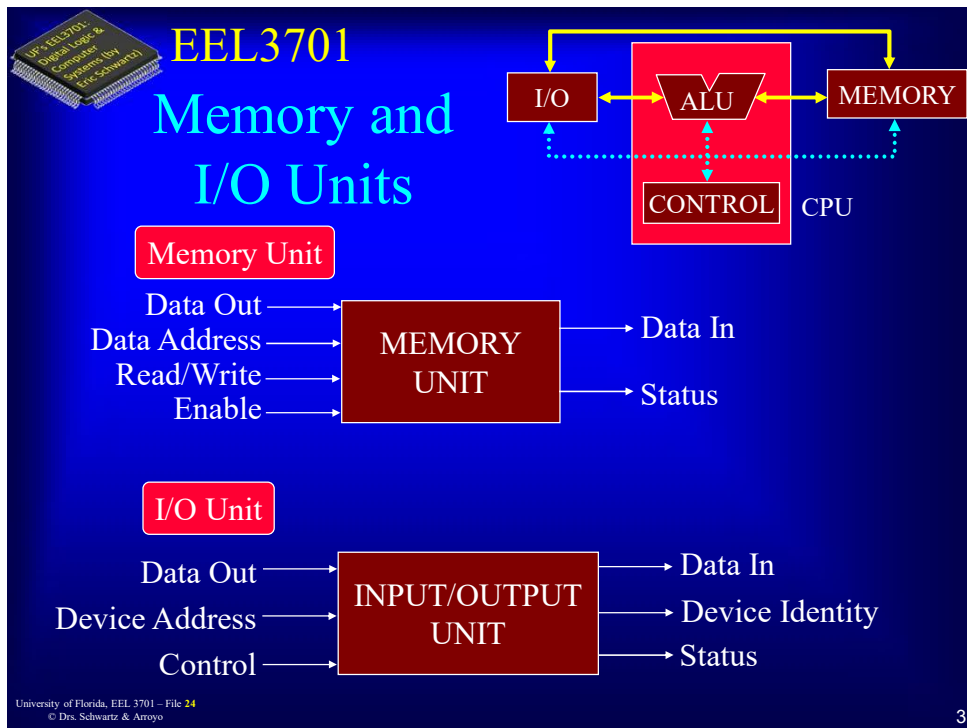
Look into my ...

See examples on web:
DirAddr.asm, ExtAddr.asm, IndAddr.asm, ImmAddr.asm, Phone.asm

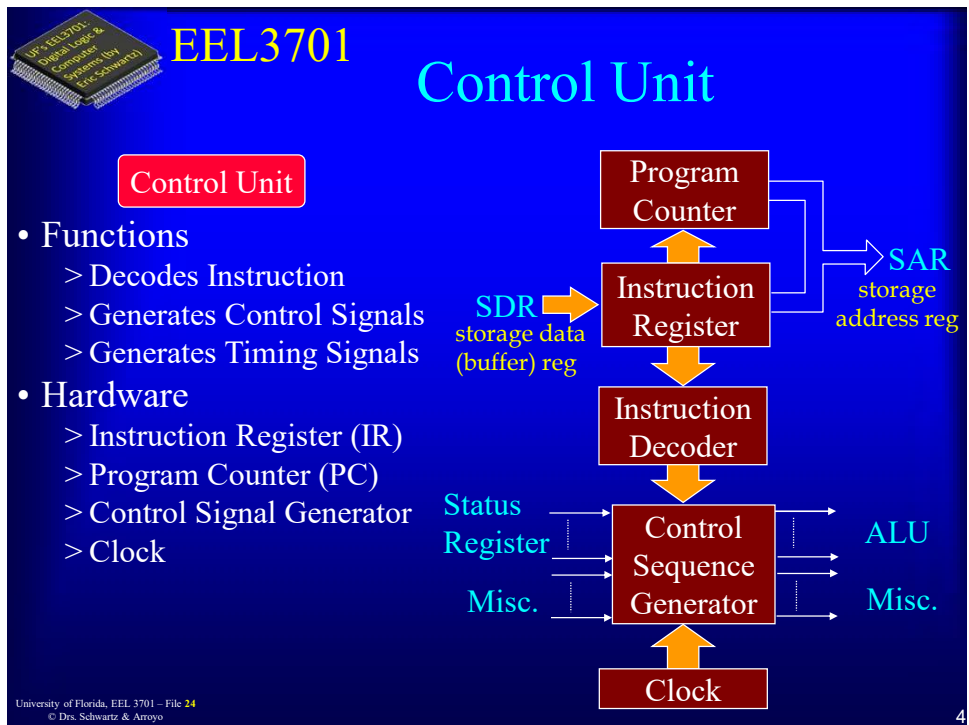University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

1

1

EEL3701 Computer Functional Block Diagram

```
I/O &          ALU          MEMORY
Peripherals

            CONTROL
              CPU
```

——— DATA
——— ADDRESS
·········· CONTROL

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

2

2

GCPU, Comp Org, 68HC11, Assembly

EEL3701 Memory and I/O Units

Memory Unit

Data Out →
Data Address →
Read/Write →
Enable →

MEMORY UNIT

→ Data In
→ Status

I/O Unit

Data Out →
Device Address →
Control →

INPUT/OUTPUT UNIT

→ Data In
→ Device Identity
→ Status

I/O ←→ ALU ←→ MEMORY

CONTROL   CPU

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

3

3

EEL3701 Control Unit

Control Unit

• Functions
> Decodes Instruction
> Generates Control Signals
> Generates Timing Signals
• Hardware
> Instruction Register (IR)
> Program Counter (PC)
> Control Signal Generator
> Clock

Program Counter

SDR
storage data
(buffer) reg

Instruction Register

SAR
storage address reg

Instruction Decoder

Status Register

Control Sequence Generator

Misc.

ALU

Misc.

Clock

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

4

4

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

2

EEL3701
Arithmetic/Logic Unit (ALU)

Arithmetic/Logical Unit (ALU)

• Functions
  > Arithmetic operations on data
  > Logical operations on data
  > Shifting operations on data
  > Status checking on results
• Hardware
  > Arithmetic/Logical circuits
  > Accumulator
  > Shifter
  > Status Register

Shift/No Shift
Left/Right   } from
Logical/Arithmetic } C.U.

Shifter

Status Register

Flags to Control Unit

ALU

Function Code (from Control Unit)

Accumulator

To Memory   From Memory
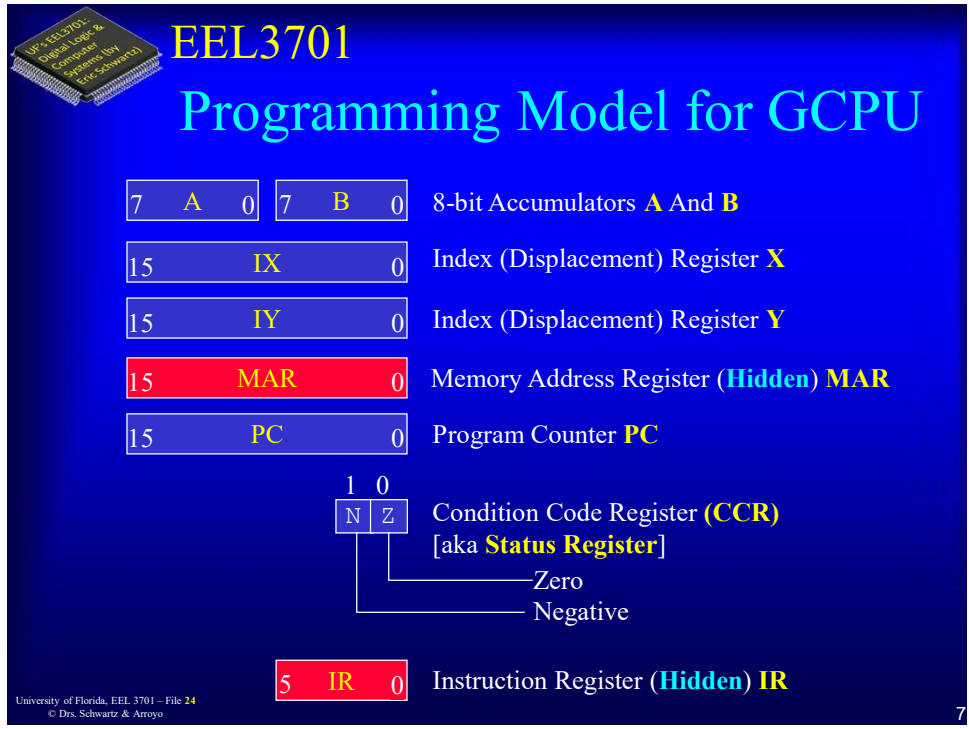
5

EEL3701
Instruction Register

• The n-bit instruction register consists of a MUX and a D-FF.
  > The MUX has a select line, IR_LD
  > The output of the MUX goes to the D input of the D-FF
  > The output of the D-FF is the 0 input of the MUX
  > The 1 input of the MUX comes from the input bus

6

EEL3701 — Programming Model for GCPU

| 7 | A | 0 | 7 | B | 0 | 8-bit Accumulators **A** And **B** |

| 15 | IX | 0 | Index (Displacement) Register **X** |

| 15 | IY | 0 | Index (Displacement) Register **Y** |

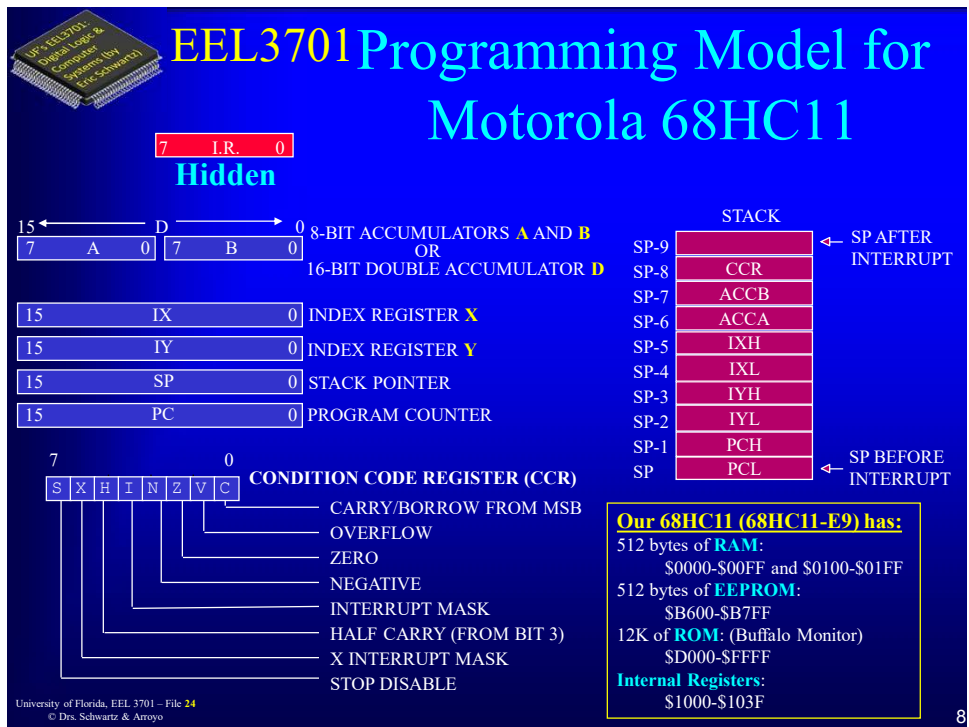| 15 | MAR | 0 | Memory Address Register (**Hidden**) **MAR** |

| 15 | PC | 0 | Program Counter **PC** |

1 0
| N | Z | Condition Code Register **(CCR)**
[aka **Status Register**]
—————— Zero
—————— Negative

| 5 | IR | 0 | Instruction Register (**Hidden**) **IR**

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

7

7

EEL3701 Programming Model for Motorola 68HC11

| 7 | I.R. | 0 |
Hidden

| 15 | D | 0 | 8-BIT ACCUMULATORS **A** AND **B**
| 7 | A | 0 | 7 | B | 0 | OR
16-BIT DOUBLE ACCUMULATOR **D**

| 15 | IX | 0 | INDEX REGISTER **X**
| 15 | IY | 0 | INDEX REGISTER **Y**
| 15 | SP | 0 | STACK POINTER
| 15 | PC | 0 | PROGRAM COUNTER

7   0
| S | X | H | I | N | Z | V | C | CONDITION CODE REGISTER (CCR)
—————— CARRY/BORROW FROM MSB
—————— OVERFLOW
—————— ZERO
—————— NEGATIVE
—————— INTERRUPT MASK
—————— HALF CARRY (FROM BIT 3)
—————— X INTERRUPT MASK
—————— STOP DISABLE

STACK
| | SP AFTER
| SP-9 | | INTERRUPT
SP-8	CCR
SP-7	ACCB
SP-6	ACCA
SP-5	IXH
SP-4	IXL
SP-3	IYH
SP-2	IYL
SP-1	PCH
SP	PCL	SP BEFORE
                INTERRUPT

Our 68HC11 (68HC11-E9) has:
512 bytes of **RAM**:
    $0000-$00FF and $0100-$01FF
512 bytes of **EEPROM**:
    $B600-$B7FF
12K of **ROM**: (Buffalo Monitor)
    $D000-$FFFF
Internal Registers:
    $1000-$103F

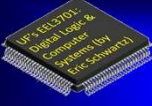University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

8

8
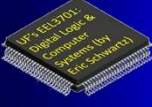
EEL3701

Schwartz in Jamaica

EEL3701

General-Purpose Registers (GCPU & 68HC11)

- There are two general-purpose registers. They are referred to as 8-bit registers **A** and **B**.
- Registers A and B, often called *accumulators*, are the most important data registers. A and B can store **8-bit** numbers.
- Examples:

LDAA VALUE1	; Move the byte at location VALUE1 to Register A.
LDAB VALUE2	; Move the byte at location VALUE2 to Register B.
SUM_BA	; Add the byte in Register B to A, the sum replaces
*	; the content of Register A. (**68HC11 spelling is ABA**)
SHFA_L	; Shift the contents of Register A to the left by 1 bit.
*	; (**68HC11 spelling is LSLA or ASLA**)

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo
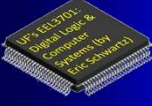
EEL3701 — General-Purpose Registers (68HC11)

- Registers **A** and **B** are sometimes treated together as a single 16-bit register **D** in some instructions.
- Examples:

  LDD   WORD1   ; The 16-bit word at location WORD1 is moved to Register D.
  *   (A number is stored in A and B treated as 16-bit register D).
  ADDD WORD2   ; The 16-bit word at location WORD2 is added to Register D.

- Most (but not all) of the instructions that involve an 8-bit register can use either A or B as one of the instruction operands. The instruction set of the 68HC11 is said to be ***nearly symmetric***. Symmetric instruction sets are programmer friendly in that the programmer need only remember a single instruction mnemonic.
- Examples:

  LDA (LDAA, LDAB), STA (STAA, STAB), ROL (ROLA, ROLB), etc.

University of Florida, EEL 3701 – File **24**
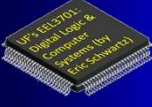© Drs. Schwartz & Arroyo
11

11

EEL3701 — GCPU Special-Purpose Registers

- **Condition Code Register (CCR)**: A 2-bit flag register in which condition codes (binary flags) are stored and tested. Also called the **Status Register**.
- **Index Registers (IX and IY)**: The 16-bit registers used to store the index value for operands retrieved using the indexed addressing mode. Also called **X** and **Y**.
- **Program Counter (PC)**: A 16-bit register whose content addresses the memory location that contains the next instruction to be executed.
- **Memory Address Register (MAR)**: A 16-bit register which contains the address of the memory location to be referenced. Used for **Extended Addressing** instructions.

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo
12

12

EEL3701   Assembly Language Programming Syntax

- For the microprocessor to understand our commands, each instruction must conform to a specific format.
  - > The general format for an GCPU/68HC11 instruction:

    [Label:]   Operation-Mnemonic   Operand (s)   Comments
- A **label** is a symbolic name for the address of an instruction.
  - > Most branch instructions utilize a label as the symbolic name of an address to which a jump may occur.
  - > When associated with an executable instruction, a label is separated from the operand by at least one whitespace character (blanks, tabs, etc.), or optionally by a colon (:).
  - > If the colon is used, it is not part of the label but merely acts to set the label off from the rest of the line.

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo   13

13
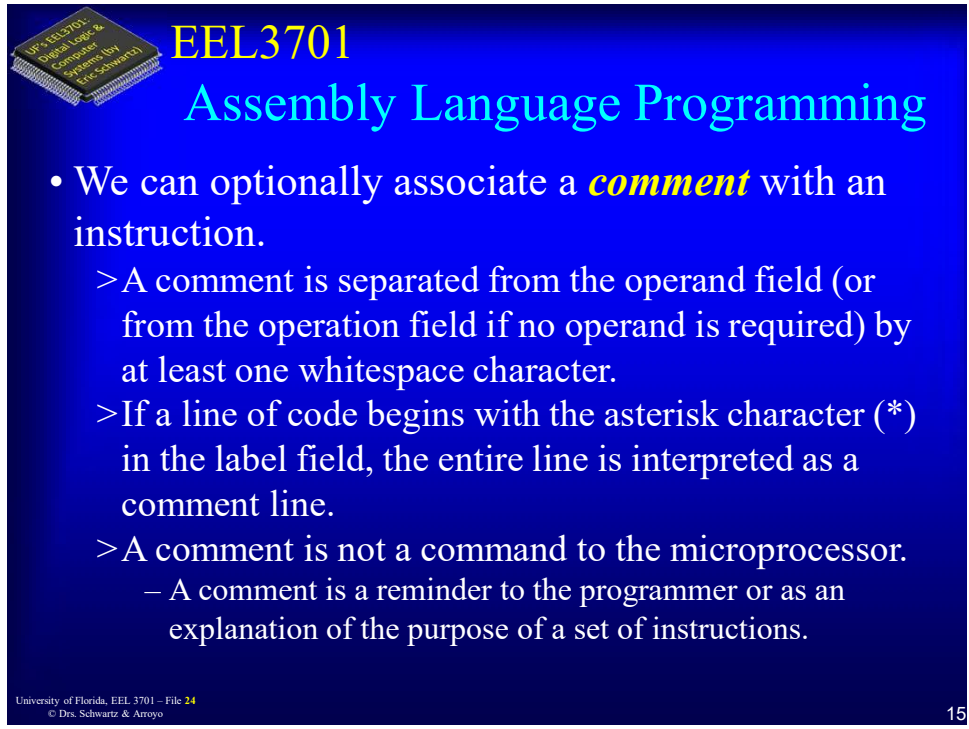
EEL3701   Assembly Language Programming

- Each instruction must include, the *operation* that the microprocessor is to perform.
  - > Each operation code is an abbreviation (*mnemonic*) of the corresponding command.
- Each instruction may also include an *operand* (or *operands*) that is the object of the operation.
  - > An operand can be a memory location (source or destination address), an external memory-mapped register, a label, a numeric value, a register-indexed address, etc.
  - > Depending on the operation, an instruction can have zero, one, two, three operands.

University of Florida, EEL 3701 – File **24**
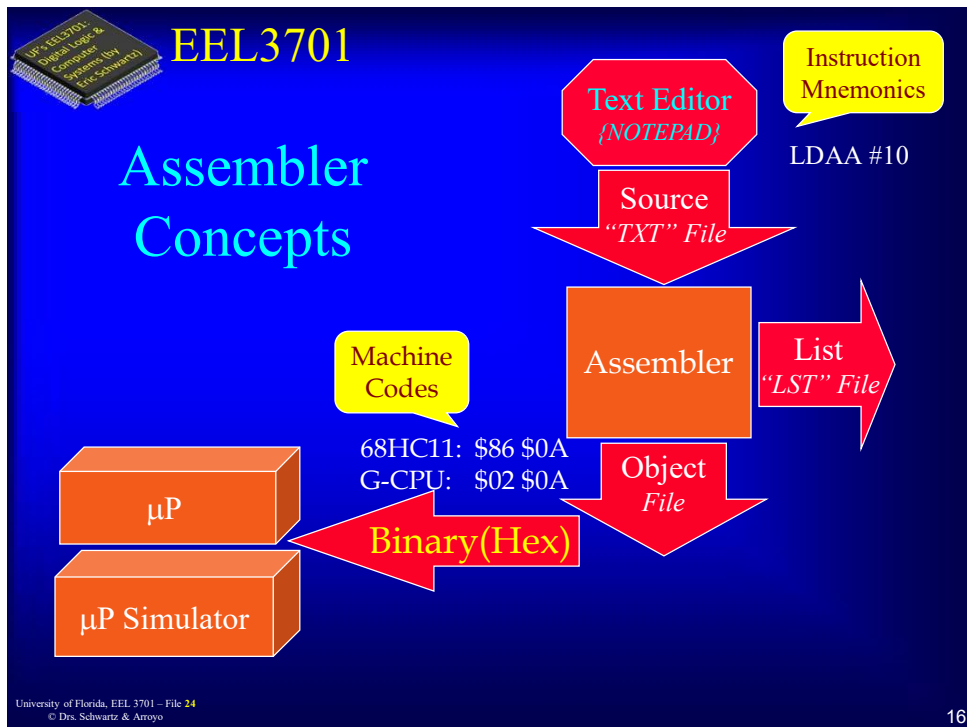© Drs. Schwartz & Arroyo   14

14

EEL3701
Assembly Language Programming

- We can optionally associate a ***comment*** with an instruction.
  - >A comment is separated from the operand field (or from the operation field if no operand is required) by at least one whitespace character.
  - >If a line of code begins with the asterisk character (*) in the label field, the entire line is interpreted as a comment line.
  - >A comment is not a command to the microprocessor.
    - – A comment is a reminder to the programmer or as an explanation of the purpose of a set of instructions.

15

15

EEL3701

Assembler Concepts

Instruction Mnemonics

Text Editor
{NOTEPAD}

LDAA #10

Source
"TXT" File

Machine Codes

Assembler

List
"LST" File

68HC11: $86 $0A
G-CPU: $02 $0A

Object
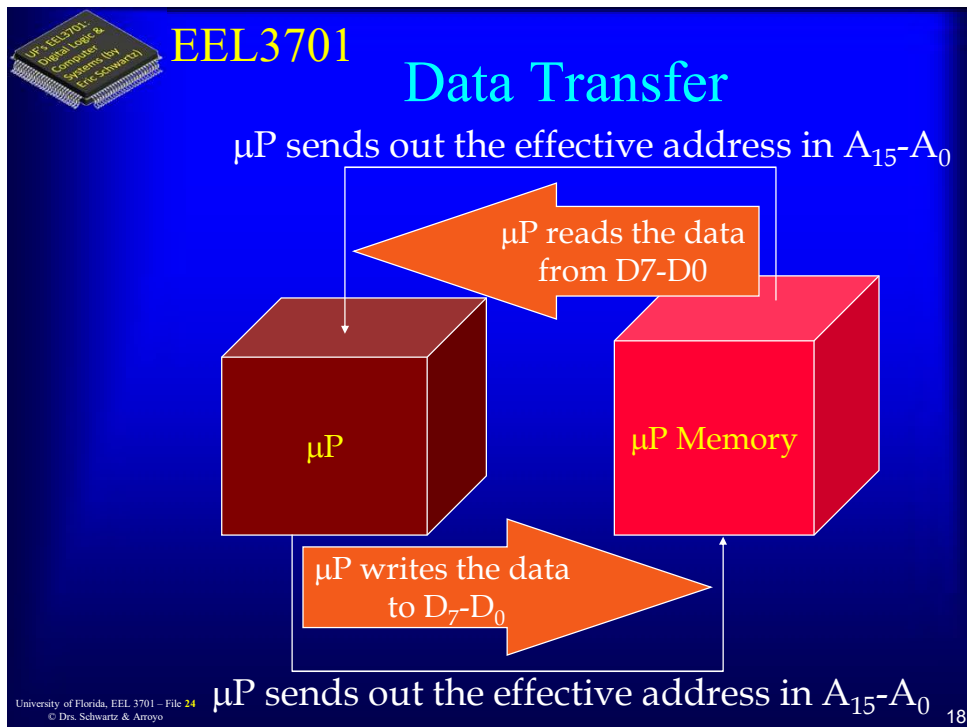*File*

Binary(Hex)

μP

μP Simulator

16

16

EEL3701
Assembly Language, Machine Language, and Program Assembly
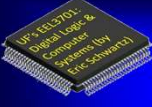
- A set of assembly language instructions is called an *assembly language program* and is also know as *source code*.
  - > An assembly language program must be transformed into a form that can be stored in memory locations, and understood by the microprocessor.
  - > This form is called *machine language program*, also called *object code*.
- The transformation is called *program assembly* and is accomplished with a computer program called an *assembler*.
- The machine language program is simply a *coded version* of the assembly language program, with each machine language instruction corresponding to an assembly language instruction.

University of Florida, EEL 3701 – File **24**
   © Drs. Schwartz & Arroyo

17

17

EEL3701
Data Transfer

μP sends out the effective address in A_{15}-A_0

μP reads the data from D7-D0

μP

μP Memory

μP writes the data to D_7-D_0

University of Florida, EEL 3701 – File **24**
   © Drs. Schwartz & Arroyo

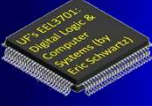μP sends out the effective address in A_{15}-A_0

18

18

EEL3701
Data Transfer Instructions

- A μP system performs all its functions through a sequence of data transfers and data transformations.
  - \> Data transfer between the internal registers within the μP.
    - – Ex: TAB, TBA
  - \> Data transfer between a memory register and an internal register within the μP.
    - – Ex: LDAA addr, LDX addr, LDAB #data
  - \> Data transfer between a peripheral devices and an internal register within the μP.
    - – Ex: LDAA memory-mapped-io, LDX memory-mapped-io

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo
19

19

EEL3701 Data Transfer Between Internal Registers
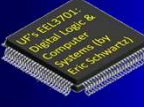
- Data transfers between internal registers use the ***inherent addressing mode*** - everything needed to execute the transfer instruction is inherently known by the CPU.
- The 8-bit transfer instructions have a general mnemonic given by T*wz*, where *w* (one of A or B) is the source register and *z* (A or B and *w* ≠ *z*) is the destination register.

(Ex) TAB: Transfer Register A to B
(BEFORE)   (AFTER)

A | 0 0 0 1 0 0 0 1   A | 0 0 0 1 0 0 0 1
B | 1 1 1 1 0 0 0 0   B | 0 0 0 1 0 0 0 1

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo
20

20

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

10

EEL3701 More Data Transfer Instructions

- Transfers between CCR and Accumulator A
- A 16-bit transfers between D (A and B together as a 16-bit register) and either Index Register X or Index Register Y.

Example: XGDX (exchange Register D with X)
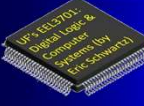
(BEFORE)

D $FF00

X $00FF

(AFTER)

D $00FF

X $FF00

68HC11
(N/A on GCPU)

Instruction	Description	Address Mode	Flags Affected
TAB/TBA	(B/A) ← (A/B)	Inherent	N,Z,V
TAP	(CCR) ← (A)	Inherent	All
TPA	(A) ← (CCR)	Inherent	None
XGDX/XGDY	(D) ↔ (X/Y)	Inherent	None

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

21

21

EEL3701 Data Transfer Between Memory & Internal Register

- During the execution of a program, data is frequently transferred, often in large quantities, between the memory locations and the internal registers. Instructions for such transfers, commonly called *memory reference instructions*, have two operands, a *source* and a *destination*, one of which may be implied. One of the two operands specifies an internal register, and the other the *effective address* of a memory location.
- Definition
  Effective Address: Where data comes from or goes to
- The manner of specifying the effective address is called the *addressing mode*. For the 68HC11, six addressing modes are possible — direct, extended, indexed, immediate, relative and the inherent mode (used for register-to-register transfers).

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

22

22

EEL3701   **Direct** Addressing Mode (68HC11)

Effective Address in A_{15}-A_0

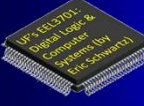| 0000	0000	User-Supplied	User-Supplied

µP Supplied

Example: LDAA $37   ; Pg 0, Locn $37 (A=$0037)

or   LDAA $0037   ; also Pg 0, Locn $37

0000	0000	0011	0111

|← Page Number $00 →|

23

23
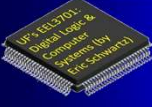
EEL3701   **Direct** Addressing Mode (68HC11)

- **Direct addressing** allows the user (through the assembler) to access Memory Locations $0 through $FF using only the least significant byte of the 16-bit memory location that is to be referenced.
  - > The high order byte of the effective address is assumed to be $00 ($00_{16}$) and is not included with the instruction operation code when the program is executed by the µP.
- An advantage is that execution time is reduced by requiring only one memory read to determine the effective address.
- Another advantage is the savings of one byte in program memory.
- The limitation is that it restricts the use of direct addressing mode to operands in the $0000-$00FF area of memory (called the direct page or page 0).
  - > Thus, direct addressing in this 256-byte area should be reserved for frequently referenced data, or for program code which requires high-speed execution. The direct addressing mode is sometimes called the **zero-page addressing mode**.

24

24

EEL3701 **Direct** Addressing Examples (68HC11)

• Load Instruction Using Direct Addressing Mode

> LDAA: Load Register A / LDAB: Load Register B

1.   LDAB $10   ;Load the 8-bit value in 001016 into Register B
2.   LDAB %00010000   ;Same as above
3. TESTV EQU $10   ;TESTV has value $10
            LDAB TESTV   ;Also loads the 8-bit value in memory location
      *   ; $0010 (also named TESTV) into Register B
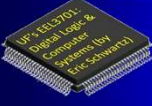
MEMORY   Addresses   See example

B $?? $000E
   BEFORE $000F
                 $28 $0010   DirAddr.asm
B $28 $0011
   AFTER

          $00FF

University of Florida, EEL 3701 – File **24**
      © Drs. Schwartz & Arroyo   25

25

EEL3701 **Direct** Addressing Examples (68HC11)

• Store Instruction Using Direct Addressing Mode

> STAA: Store Register A /STAB: Store Register B

1.   STAB $11   ;Store the 8-bit value in Register B into
      *   ; memory location $0011
2.   STAB %00010001   ;Also stores the 8-bit value in Register B into
      *   ; memory location %00010001
3. LEVEL EQU $11   ;LEVEL has value $11
            STAB LEVEL   ; Also stores the 8-bit value in Register B into
      *   ; memory location $11 (also named LEVEL)

B $77   B $77
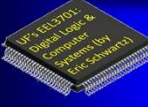   MEMORY BEFORE   MEMORY AFTER

              Addresses   Addresses
              $000E $000E
              $000F $000F
    $28 $0010 $28 $0010
    $?? $0011 $77 $0011

University of Florida, EEL 3701 – File **24**
      © Drs. Schwartz & Arroyo   26

26

University of Florida, EEL 3701 – File **24**
      © Drs. Schwartz & Arroyo   13

EEL3701
Extended Addressing Mode

Effective Address in A_{15}-A_0

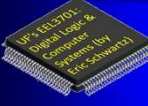| User-Supplied	User-Supplied	User-Supplied	User-Supplied

Example: LDAA $1234 ; Pg $12, Locn $34

or   LDAA $AB34 ; Pg $AB, Locn $34

0001	0010	0011	0100

|← Non-zero Page No. $12 →|

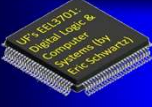University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo
27

27

EEL3701
Extended Addressing Mode

- In the ***extended addressing*** mode, we specify, as part of an instruction, the entire 16-bit memory location that is to be referenced. Extended addressing allows the programmer to reference any location in the entire memory range of the GCPU. Since addresses are 16-bit quantities, the range of valid memory references is 000016-FFFF16. The instruction includes as part of the machine code the complete 2-byte address of the operand.
- Example: The first line below performs direct addressing; the second line below performs extended addressing for the 68HC11
      LDAB $10   ;68HC11 machine codes: d6 10
      LDAA $4237 ;68HC11 machine codes: b6 42 37
- Example: The below lines perform extended for GCPU since the GCPU does **not** have direct addressing; note the order of address bytes.
      LDAB $10   ;G-CPU machine codes: 05 10 00
      LDAA $4237 ;G-CPU machine codes: 04 37 42

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo
28

28

EEL3701
Little-Endian and Big-Endian

- Little-Endian: Describes a computer architecture in which, within a given 16-bit word, bytes at lower addresses have lower significance (the word is stored `little-end-first').
  - \> The PDP-11 and VAX families of computers and Intel microprocessors and a lot of communications and networking hardware are little-endian. (GCPU)
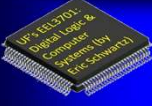- Ex: LDAA $4237 ;**G-CPU** machine codes: **04 37 42**

GCPU

3AB0	04
3AB1	37
3AB2	42

- Big-Endian: Describes a computer architecture in which, within a given multi-byte numeric representation, the most significant byte has the lowest address (the word is stored `big-end-first').
  - \> IBM 370 family, the PDP-10, the Motorola microprocessor families, and most RISC designs are big-endian. (68HC11)
- Ex: LDAA $4237 ;**68HC11** machine codes: **b6 42 37**

68HC11

3AB0	B6
3AB1	42
3AB2	37

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

29

29

EEL3701
<u>Extended</u> Addressing Examples

- ☐ Load Instruction Using Extended Addressing Mode
  - LDAA: Load Register A /LDAB: Load Register B
    1.   LDAB @420   ;Load a 8-bit value in 420_8 (0110_{16}) into Register B
    2.   LDAB 272   ;the same as above. 272_{10} (0110_{16})
    3. DATA EQU $0110   ;DATA has value $0110
            LDAB DATA   ;Also loads the 8-bit value in Memory Location
       *   ; 0110_{16} (also named DATA) into Register B

MEMORY | Addresses

B	$??
	BEFORE

⇩

B	$28
	AFTER

	Addresses
	$010E
	$010F
$28	$0110
	$0111
⋮	
	$FFFF

ExtAddr.asm

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

30

30

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

15

EEL3701
<u>Extended</u> Addressing Examples

☐ Store Instruction Using Extended Addressing Mode
  • STAA: Store Register A /STAB: Store Register B
    1.   STAB @421   ;Store the 8-bit value in Register B into
       *   ; memory location 421_8 ($100\,010\,001_2 = 0111_{16}$)
    2.   STAB 273   ;Also stores the 8-bit value in Register B into
       *   ; memory location 273_{10} (0111_{16})
    3. LPT1 EQU $0111   ;LPT1 has value $0111
           STAB LPT1   ;Also stores the 8-bit value in Register B into
       *   ; memory location 0111_{16} (also named LPT1)

B [$77]
MEMORY BEFORE
:

ExtAddr.asm

Addresses
$010E
$010F
[$28] $0110
[$??] $0111
:

B [$77]
MEMORY AFTER
:

Addresses
$010E
$010F
[$28] $0110
[$77] $0111
:

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

31

31

EEL3701
<u>Indexed</u> Addressing Mode
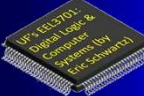
Example: Assume IX = $1234, then
LDAA $2A,X   ; Loads the content of Page $12 Locn $34+$2A

IX or IY	User-Supplied	User-Supplied	User-Supplied	User-Supplied
	0001	0010	0011 +	0100

1 Byte Displacement
(8-bit unsigned number)

User-Supplied	User-Supplied
0010	1010

Effective Address in A_{15}-A_0

0001	0010	0101	1110

But IX / IY remain unchanged!!!

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

32

32

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

16

EEL3701 **Indexed** Addressing Mode

- With *indexed addressing* we do not directly specify the effective address as part of an instruction. Instead, we specify one of two index registers (Index Register X or Y) that contain an address which is within 255 bytes of the 16-bit operand address. We can think of the value stored in the index register as the base address used in calculating the actual effective address by the following formula:

  (effective address) = (base address [value in X or Y])
  + (8-bit unsigned offset/displacement)

- This addressing mode allows referencing any memory location in the address space. It is used primarily for manipulating contiguous memory locations (a linear array or vector of memory addresses).

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo
33

33

EEL3701
Indexed Addressing Example

- [Example] Retrieve a 7-digit telephone number (846-1509) stored in the memory as a 7 BCD digit sequence starting at Memory Location 0010_{16}
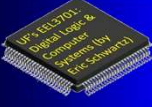
```
X   [        $0010        ]

    LDAA  0,X
    STAA  DIGIT1

    LDAA  1,X
    STAA  DIGIT2

       :

    LDAA  5,X
    STAA  DIGIT6

    LDAA  6,X
    STAA  DIGIT7
```

MEMORY

$08	$0010
$04	$0011
$06	$0012
$01	$0013
$05	$0014
$00	$0015
$09	$0016
???	$0017

Repeat with a better program. This time use a single address for the "Dialer" and use a loop.

phone.asm

University of Florida, EEL 3701 – File **24**
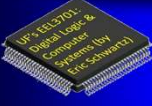© Drs. Schwartz & Arroyo
34

34

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

17

EEL3701
Indexed Addressing

☐ **An advantage of using indexed addressing** becomes apparent if we wish to repeat the telephone example using a new number stored at a different memory location, say, Memory Location $202A. One need only reload Index Register X with its new value ($202A) and we can use the same 14 instruction sequence in Fig. 9.8.

☐ If no offset is specified or desired, the instruction generated by the assembler will have $00 in the offset byte.

☐ **The offset is an unsigned byte** (an 8-bit binary number) that when added to the current value of the index register, yields the effective address of the operand leaving the index register unchanged.

☐ Because the offset is unsigned, a negative offset cannot be specified.

☐ Offsets range from 0 ($00) to 255 ($FF) inclusive.

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo   35

35

EEL3701
Immediate Addressing Mode

Effective Address in A_{15}-A_0

PC_{15}-PC_{12}	PC_{11}-PC_8	PC_7-PC_4	PC_3-PC_0

µP Supplied

Example:

LDAA   #$34   ; Put $34 inside Reg. A

• Action: The number $34 is placed after the LDAA opcode imbedded in the program code. This results in the *number* $34 being loaded into Register A.

• The effective address is the address of the immediate number.

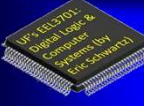University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo   36

36

EEL3701 LDAA Examples with Various Addressing Modes (GCPU)

- **Show with memory maps**

$0101	LDAA	#$34	→

EA = **$0102**

$0101	$02
$0102	$34
$0103	

$5020	LDAA	$AB34	

EA = **$AB34**

$5020	$04
$5021	$34
$5022	$AB

$A32B	LDY	#$C007	→
$A32E	LDAA	$F0,Y	

EA = **$A32C**

EA = **$C007+$F0 = $C0F7**

$A32B	$09
$A32C	$07
$A32D	$C0
$A32E	$0D
$A32F	$F0
$A330	
$A331	

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

37

37

EEL3701
__Immediate__ Addressing Mode

- In the ***immediate addressing*** mode, the instruction contains the data itself, as an operand. The data can be an 8-bit quantity (a byte), or a 16-bit quantity (a word), depending on the instruction or the destination of the quantity. An immediate operand is indicated by the character # used as a prefix for a numeric operand expression.
- A variety of symbols and expressions can be used following the character # sign (and sometimes without the # sign too)
  - (none) : decimal quantities (the default base)
  - $: hexadecimal quantities
  - @   : octal quantities
  - %   : binary quantities
  - '   : a single ASCII character

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

38

38

EEL3701 Immediate Addressing Examples

```
          ORG     $0010        ;The program segment begins at location 0010₁₆.
                               ;Symbol START is implicitly equated to 0010₁₆.
START    LDAA    #22          ;Load 22 into Register A
          LDAB    #$34         ;Load 34₁₆ into Register B
CAT      EQU     7            ;Symbol CAT is equated to 7
          LDAA    #CAT         ;Load 7 into Register A
          LDD     #$1234       ;Load 1234₁₆ into Register D
          LDY     #$B100       ;Load B100₁₆ into Register Y
          LDX     #START       ;Load 0010₁₆ into Register X
```

$START = 0010_{16}$

ImmAddr.asm

CAT = 7

The value of a symbol that appears in the label field of an EQU directive is defined by the value in the operand field of the statement.

The value of any symbol is equal to its address **except** when used in the label field of EQU statement.

EEL3701 Inherent Addressing Example

☐ *Inherent Addressing* Mode improves efficiency

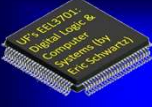   (ex) TAB, SUM_BA

(a) Example 1:   SUM_BA   ;A ← A + B

	(Before)			(After)
A	$17	→	A	$C8
B	$B1		B	$B1

```
C=0 H=0
Reg A        00010111
Reg B       +10110001
(new) Reg A  11001000
      Z=0              N=1
              V=0
                   $C8
```

EEL3701
Addressing Modes

Q: Can you determine the *Effective Address* for each of the addressing modes?

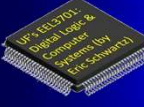A: If not, please learn ASAP. This is very important!

41

EEL3701
Common Assembler Directives

- Assembly Control
  - \>ORG   Origin (address) for next line in assembly program
- Symbol Definition
  - \>EQU   Assign permanent value
- Data Definition/Storage Allocation
  - \>DC.B Define constant byte
  - \>DC.W Define constant word
  - \>DS.B Define storage bytes
  - \>DS.W Define storage word

42

EEL3701 Assembler Directives (Pseudo-instructions)

• **ORG (Origin)**: It can be used to alter the location counter by setting it to any memory location in memory.

ORG   operand   (where operand is a 16-bit address or an expression that evaluates to a 16-bit address.)

Example:
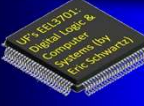* Anything under this assembler directive will begin filling up
* memory at address $7300
        ORG $7300

43

43

EEL3701 Assembler Directives (Pseudo-instructions)

• **EQU (Equate):** It informs the assembler to equate the specified symbol name to the value of the operand. In other words, when the symbolic name is subsequently encountered in the assembly process, the assembler replaces it with the binary value of the corresponding operand. The operand can be either a value or an expression that can be evaluated. It should be used to improve the clarity and readability of the assembly program.
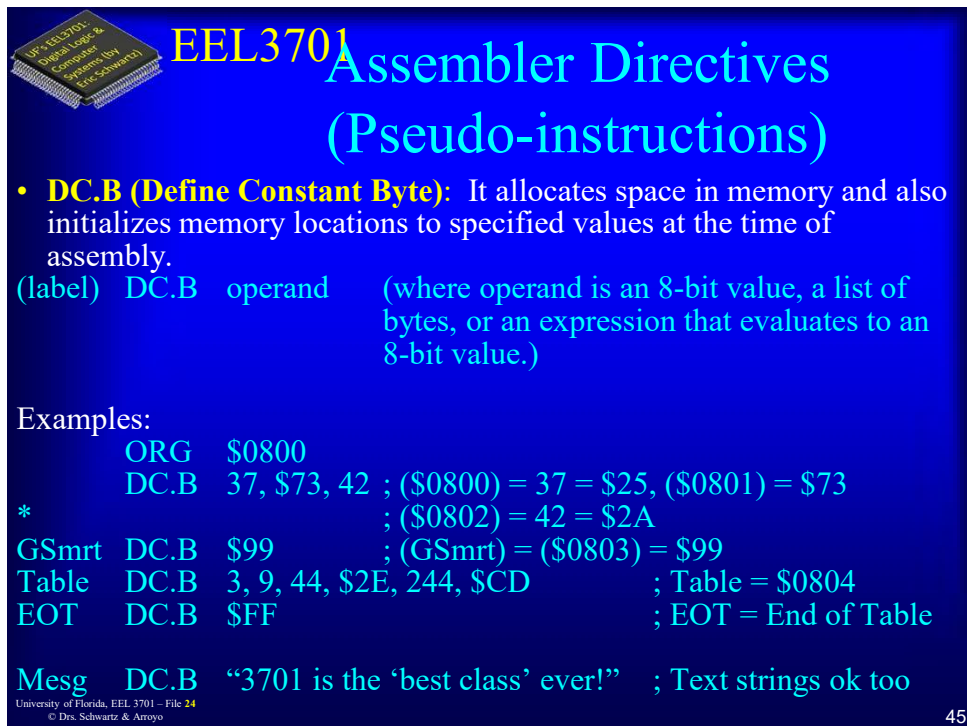
Name   EQU   operand   (where operand is a value or an expression that evaluates to a value.)

Examples:
PI   EQU   3   ; Pi will be replace everywhere in the file
*   ;   with the number 3
BestNo EQU $37   ; BestNo will be replaced by $37

44

44

EEL3701 Assembler Directives (Pseudo-instructions)

- **DC.B (Define Constant Byte)**: It allocates space in memory and also initializes memory locations to specified values at the time of assembly.

(label)   DC.B   operand   (where operand is an 8-bit value, a list of bytes, or an expression that evaluates to an 8-bit value.)
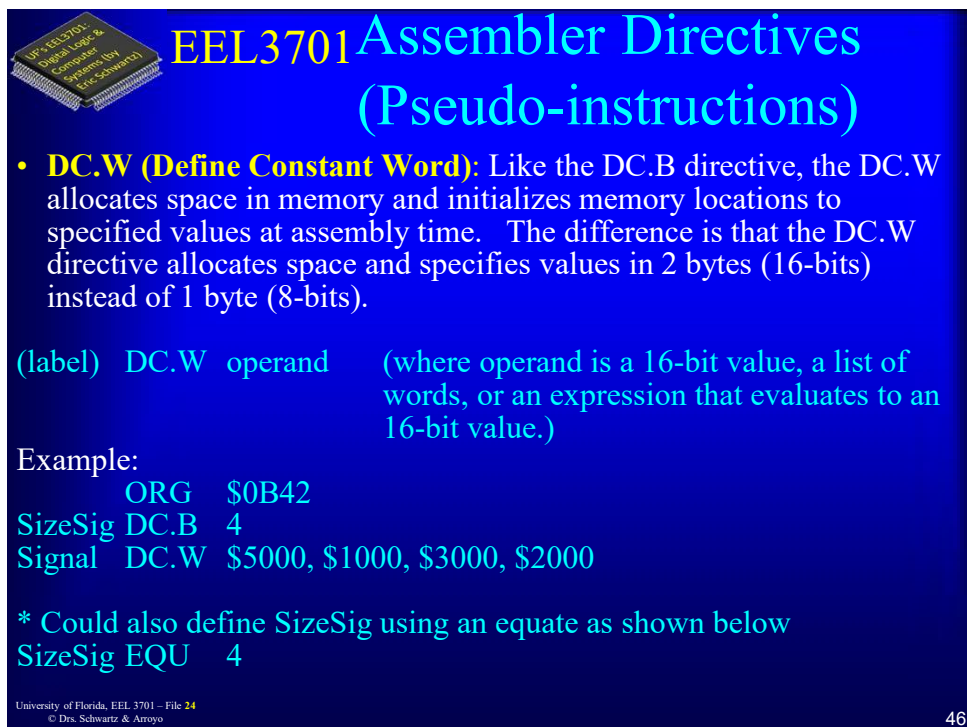
Examples:
```
        ORG    $0800
        DC.B   37, $73, 42  ; ($0800) = 37 = $25, ($0801) = $73
*                           ; ($0802) = 42 = $2A
GSmrt   DC.B   $99          ; (GSmrt) = ($0803) = $99
Table   DC.B   3, 9, 44, $2E, 244, $CD      ; Table = $0804
EOT     DC.B   $FF                          ; EOT = End of Table

Mesg    DC.B   "3701 is the 'best class' ever!"   ; Text strings ok too
```
University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo   45

45

EEL3701 Assembler Directives (Pseudo-instructions)

- **DC.W (Define Constant Word)**: Like the DC.B directive, the DC.W allocates space in memory and initializes memory locations to specified values at assembly time.   The difference is that the DC.W directive allocates space and specifies values in 2 bytes (16-bits) instead of 1 byte (8-bits).

(label)   DC.W operand   (where operand is a 16-bit value, a list of words, or an expression that evaluates to an 16-bit value.)

Example:
```
        ORG    $0B42
SizeSig DC.B   4
Signal  DC.W   $5000, $1000, $3000, $2000

* Could also define SizeSig using an equate as shown below
SizeSig EQU    4
```
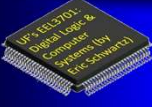University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo   46

46

EEL3701 Assembler Directives (Pseudo-instructions)

- **DS.B (Define Storage Bytes)**: It allocates a block of storage in memory, but it does not initialize the contents of the allocated memory locations. DS.B is used for variables.

(label)   DS.B   operand   (where operand is a value or an expression that evaluates to a value.)
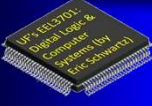
Examples:
* Space for a table is defined beginning at address $1A00 and ending
* at address $1AFF. A second table goes from $1B00-$1BFF. A single 1-byte variable is also shown.

```
        ORG   $1A00
Table   DS.B   256
Tab2    DS.B   256
Var1    DS.B   1
```

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

47

47

EEL3701 Assembler Directives (Pseudo-instructions)

- **DS.W (Define Storage Word)**: Like the DS.B directive, the DS.W allocates space in memory.   The difference is that the DS.W directive allocates space in 2 bytes (16-bits) increments instead of 1 byte (8-bits). DS.W is used for two-byte variables.

(label)   DS.W   operand   (where operand is a value or an expression that evaluates to a value.)

Examples:
* Space for a table is defined beginning at address $1A000 and ending
* at address $1A09. Next is a two-byte variable and then a table of 2 two-byte variables.

```
        ORG   $1A00
Table   DS.W   5           ; Table of 5 two-byte (word) variables
Var1    DS.W   1           ; Two-byte (word) variable
Var2    DS.W   2           ; Two two-byte (word) variables
```

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

48

48

University of Florida, EEL 3701 – File **24**
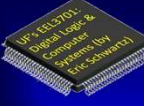© Drs. Schwartz & Arroyo

24

EEL3701
Decrement Instruction and Loops

- How can you make the equivalent of a DECA for the GCPU?
- Hint: Remember that subtraction can be accomplished by addition if a 2's complement can be calculated

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

49

49

EEL3701

The End!

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

50

50

University of Florida, EEL 3701 – File **24**
© Drs. Schwartz & Arroyo

25