



EEL3701

Menu

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



See examples on web:

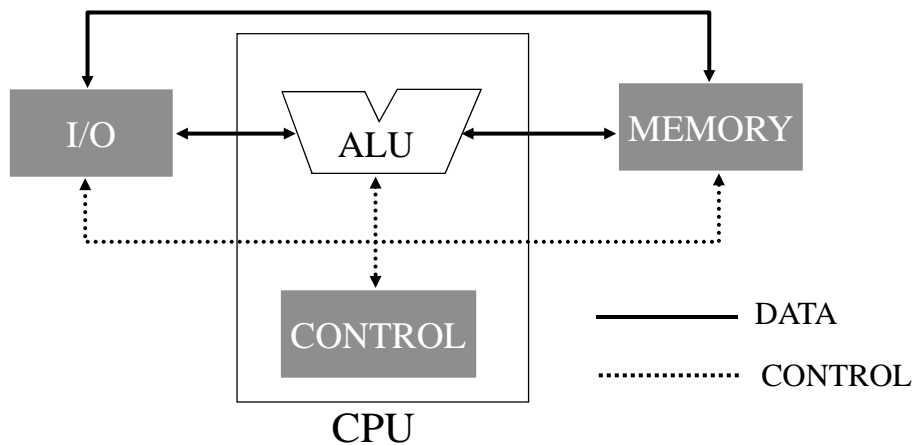
DirAddr.asm, ExtAddr.asm,
IndAddr.asm, ImmAddr.asm,
Phone.asm

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

1



EEL3701 Computer Functional Block Diagram



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

2



EEL3701

Memory and I/O Units

Memory Unit



I/O Unit

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

3

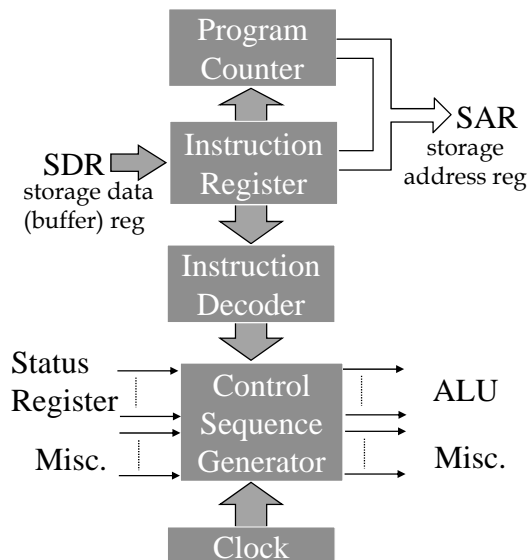


EEL3701

Control Unit

Control Unit

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

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

4



EEL3701

Arithmetic/Logic Unit (ALU)

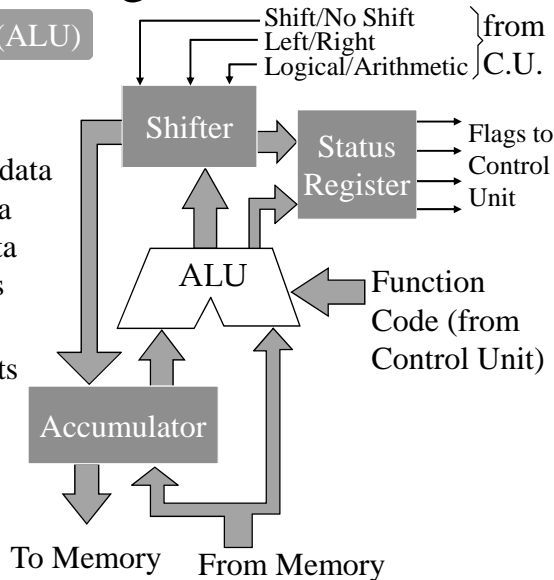
Arithmetic/Logical Unit (ALU)

- 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



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

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

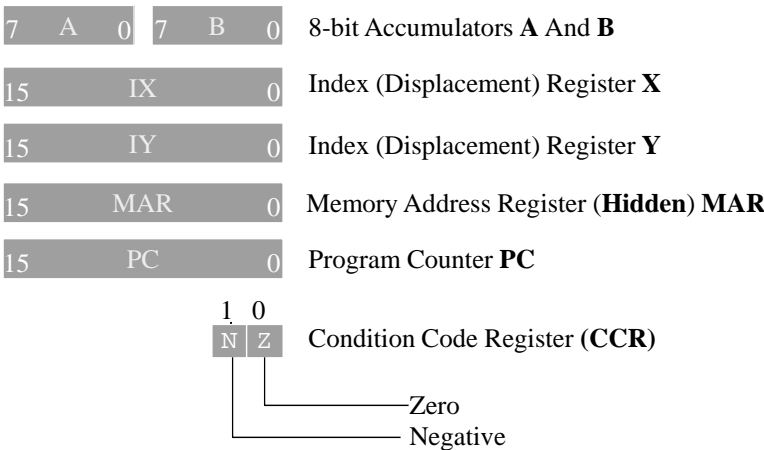
University of Florida, EEL 3701 – File 24
© Drs. Schwartz & Arroyo

6



EEL3701

Programming Model for GCPU



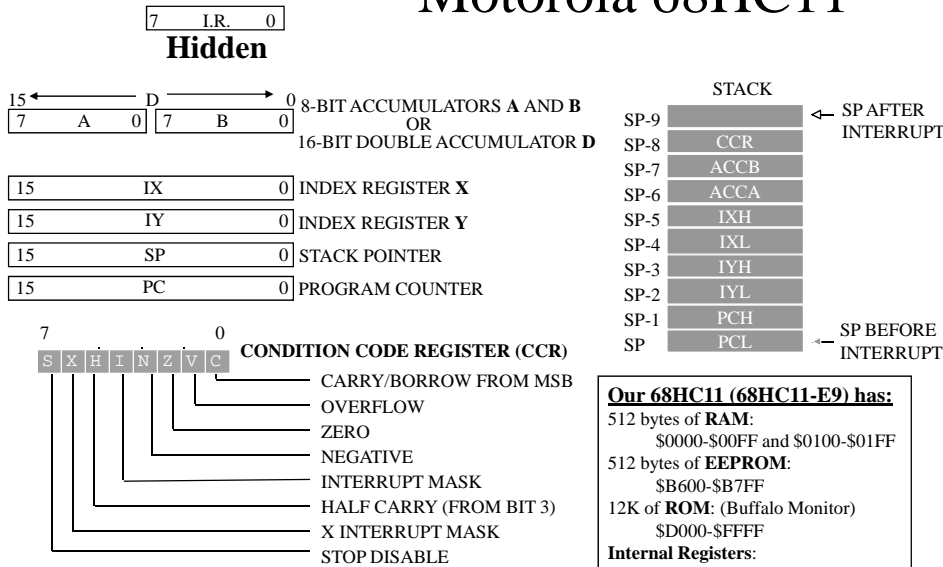
University of Florida, EEL 3701 – File 24
© Drs. Schwartz & Arroyo

7



EEL3701

Programming Model for Motorola 68HC11



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

8



EEL3701



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)
```



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

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

11



EEL3701GCPU Special-Purpose Registers

- ❖ **Condition Code Register (CCR):** A 2-bit flag register in which condition codes (binary flags) are stored and tested
- ❖ **Index Registers (IX and IY):** The 16-bit registers used to store the index value for operands retrieved using the indexed addressing mode
- ❖ **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

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

12



EEL3701

Assembly Language Programming

- ❖ Programming a microprocessor in an assembly language is somewhat like using an **OLD** electronic calculator. To use a calculator we must know what operations the calculator can perform.
- ❖ For a μP system, the set of allowed operations, called an instruction set, is a great deal more extensive than that of non-programmable calculators, and includes operations such as LOAD, SHIFT, STORE, and JUMP, in addition to ADD, SUBTRACT, MULTIPLY, DIVIDE, and so on.
- ❖ Further, the register structure of a μP system is a great deal more extensive, having different types of internal registers and external registers, as well as memory registers. To make full use of a μP system, we must know its register structure and fully understand its instruction set.

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

13



EEL3701

Assembly Language Programming

- ❖ For the microprocessor to understand our commands, each instruction must conform to a specific format.
 - < The general format for an executable 68HC11 instruction >
- | | | | |
|-----------------|---------------------------|--------------------|-----------------|
| [Label:] | Operation-Mnemonic | Operand (s) | Comments |
|-----------------|---------------------------|--------------------|-----------------|
- ❖ A *label* is a symbolic name for the address of an instruction. It is usually assigned when there is to be a branch from another instruction to that instruction or when we want to refer to the content of a specific address symbolically. 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

14



EEL3701

Assembly Language Programming

- Each instruction must include, of course, 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 one, two, three operands, or none at all.

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

15



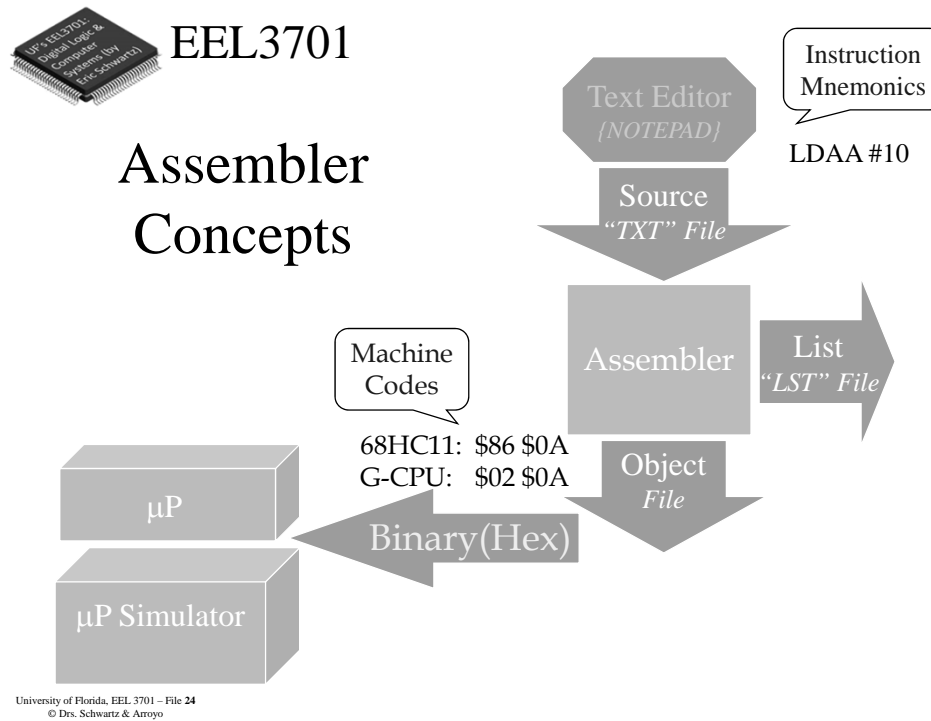
EEL3701

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. Rather, it serves simply as a reminder to the programmer or as an explanation to anyone reading the program of what the corresponding instruction does.

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

16



17

EEL3701 Machine Language and Program Assembly

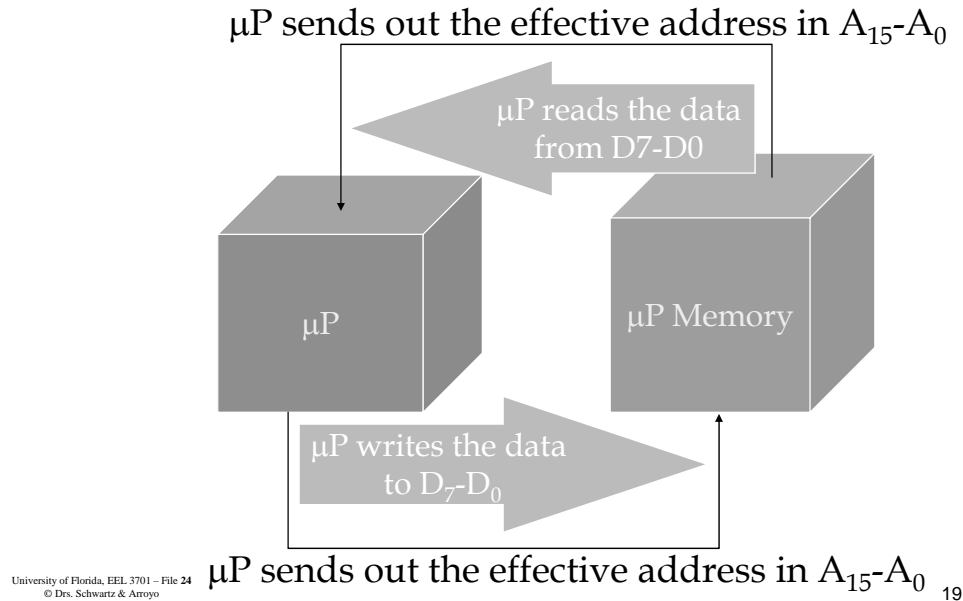
- A consequence of the stored program concept is that the assembly language instructions must be transformed into a form that can be stored in memory locations, and also that can be processed by the microprocessor. In other words, an **assembly language program**, also called **source code**, has to be transformed into a **machine language program**, also called **object code**.
- This transformation process, called **program assembly**, can be done automatically by a computer program called an **assembler** (e.g., AS11).
- 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.

18



EEL3701

Data Transfer



19



EEL3701

Data Transfer Instructions

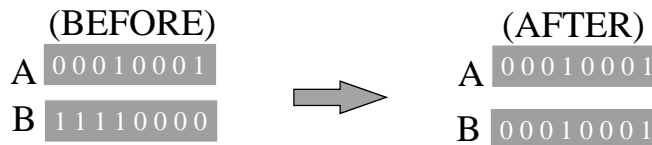
- A μP system performs all its functions through a sequence of data transfers and data transformations. The performance of a program can be greatly affected by the convenience with which the data can be transferred within the register structure of a μP system.
 - > Data transfer between the internal registers within the MCU
 - Ex: TAB, TBA
 - > Data transfer between a memory register and an internal register within the MCU
 - Ex: LDAA addr, LDX addr, LDAB #data
 - > Data transfer between a peripheral devices and an internal register within the MCU.
 - Ex: LDAA memory-mapped-io, LDX memory-mapped-io



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 Twz , where w (one of A or B) is the source register and z (A or B and $w \neq z$) is the destination register.

(Ex) TAB: Transfer Register A to B



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

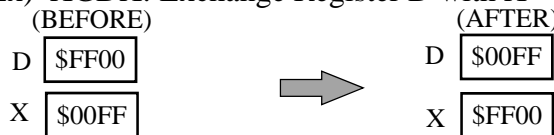
21



EEL3701 Data Transfer Between 16-bit Internal Registers

- ❖ We can also perform 16-bit transfers between D (A and B together as a 16-bit register) and either Index Register X or Index Register Y.

(Ex) XGDX: Exchange Register D with X



68HC11

Instruction	Description	Address Mode	Flags Affected
<i>8-bit Internal Register Transfers</i>			
TAB/TBA	$(B/A) \leftarrow (A/B)$	Inherent	N,Z,V
TAP	$(CCR) \leftarrow (A)$	Inherent	All
TPA	$(A) \leftarrow (CCR)$	Inherent	None
<i>16-bit Internal Register Transfers</i>			
TSX/TSY	$(X/Y) \leftarrow (SP)+1$	Inherent	None
TXS/TYS	$(SP) \leftarrow (X/Y)-1$	Inherent	None
XGDX/XGDY	$(D) \leftrightarrow (X/Y)$	Inherent	None

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

22



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

23



EEL3701 68HC11 Load and Store Instructions (GCPU similar)

Instruction	Description	Addressing Mode	Flags Affected
<i>8-bit Accumulator Load/Store</i>			
LDAA/LDAB (bopr)	(A/B) ← (addr)	Imm, Dir, Ext, Ind X, Y	N,Z,V
STAA/STAB (bopr)	(addr) ← (A/B)	Dir, Ext, Ind X, Y	N,Z,V
<i>16-bit Register Load/Store</i>			
LDw (wopr)	(w) ← (addr,addr+1)	Imm, Dir, Ext, Ind X, Y	N,Z,V
STw (wopr)	(addr,addr+1) ← (w)	Dir, Ext, Ind X, Y	N,Z,V

Notes

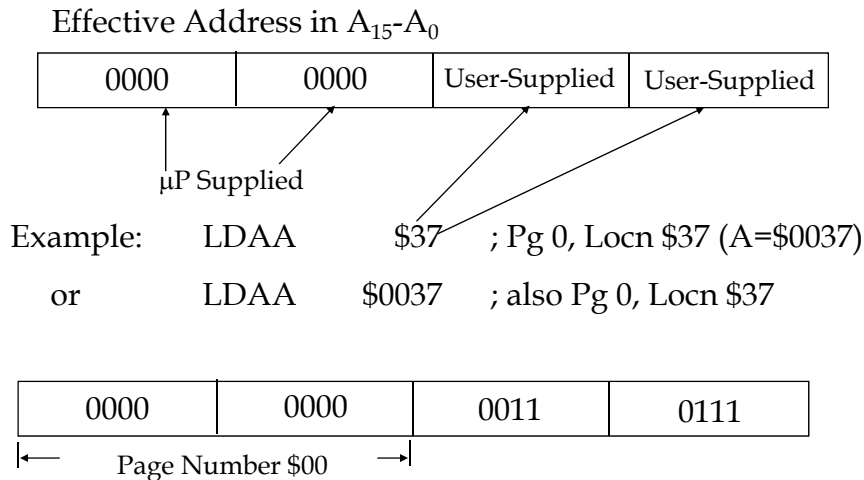
- w 16-bit Registers D, X, Y, or S (for SP)
- (bopr) Operand specification, immediate byte [#(8-bit value)], direct/extended address of a byte, 8-bit offset from X or Y.
- (wopr) Operand specification, immediate word [#(16-bit value)], direct/extended address of a word, 8-bit offset from X or Y.

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

24



EEL3701 Direct Addressing Mode (68HC11)



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

25



EEL3701 Direct Addressing Mode (68HC11)

- ❖ **Direct addressing** allows the user (through the assembler) to access Memory Locations \$0000 through \$00FF 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 MCU.
- ❖ 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 1 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**.

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

26

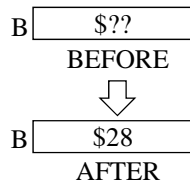


EEL3701

Direct Addressing Examples (68HC11)

❖ Load Instruction Using Direct Addressing Mode

- (a) General Format: LDAA or LDAB ;Load Register A or load Register B
 (b) LDAB \$10 ;Load the 8-bit value in 0010₁₆ into Register B.
 (c) LDAB %00010000 ;the same as above
 (d) TESTV EQU \$10 ;TESTV has value \$10
 LDAB TESTV ;Also loads the 8-bit value in Memory Location
 ;0010₁₆ (also named TESTV) into Register B.



MEMORY	Addresses
	\$000E
	\$000F
\$28	\$0010
	\$0011
⋮	
	\$00FF

See example


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

27

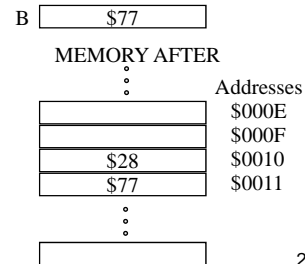
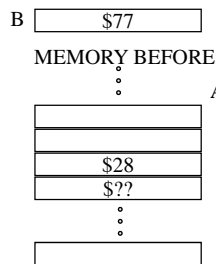


EEL3701

Direct Addressing Examples (68HC11)

❖ Store Instruction Using Direct Addressing Mode

- (a) General Format: STAA or STAB ;Store Register A or store Register B
 (b) STAB \$11 ;Store the 8-bit value in Register B into
 ; memory location 0011₁₆.
 (c) STAB %00010001 ;Also store the 8-bit value in Register B
 ; into Memory Location 00010001₂.
 (d) LEVEL EQU \$11 ;LEVEL has value \$11
 STAB LEVEL ;Also store the 8-bit value in Register B into
 ; memory location 11₁₆ (also named LEVEL).

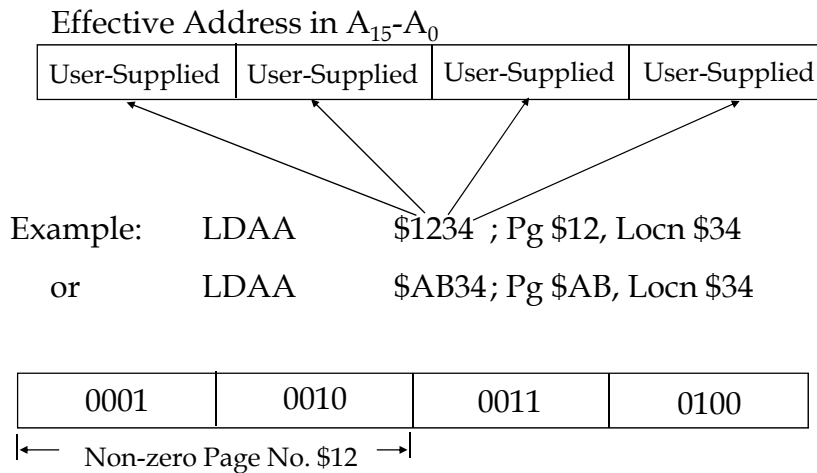
University of Florida, EEL 3701 – File 24
© Drs. Schwartz & Arroyo

28



EEL3701

Extended Addressing Mode

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

29



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 $0000_{16}-FFFF_{16}$. 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

30



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

31



EEL3701

Extended Addressing Examples

❖ Load Instruction Using Extended Addressing Mode

(a) *General Format*: LDAA or LDAB ;Load Register A or load Register B

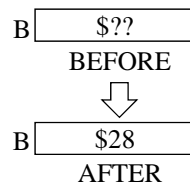
(b) LDAB @420 ;Load a 8-bit value in 420_8 (0110_{16}) into Register B.

(c) LDAB 272 ;the same as above. 272_{10} (0110_{16})

(d) 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
	\$010E
	\$010F
\$28	\$0110
	\$0111
:	
	\$FFFF

ExtAddr.asm

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

32

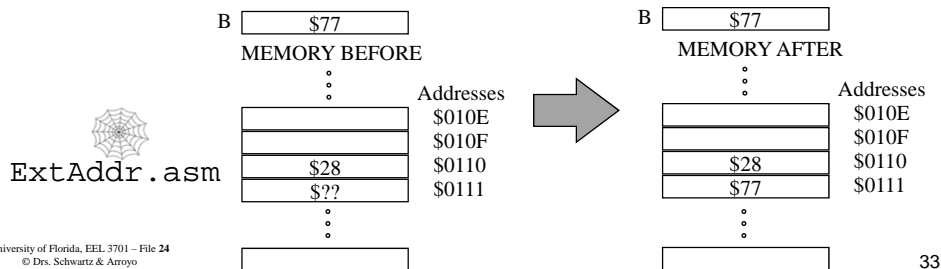


EEL3701

Extended Addressing Examples

❖ Store Instruction Using Extended Addressing Mode

- (a) General Format: STAA or STAB ;Store Register A or store Register B
 (b) STAB @421 ;Store the 8-bit value in Register B into
 * ; memory location 421_8 ($100\ 010\ 001_2 = 0111_{16}$).
 (c) STAB 273 ;Also store the 8-bit value in Register B into
 * ; memory location 273_{10} (0111_{16}).
 (d) LPT1 EQU \$0111 ;LPT1 has value \$0111
 STAB LPT1 ;Also store the 8-bit value in Register B into
 * ; memory location 0111_{16} (also named LPT1).



33

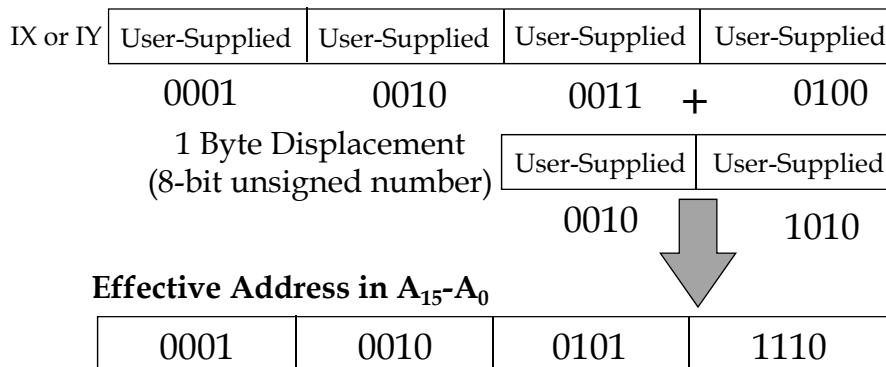


EEL3701

Indexed Addressing Mode

Example: Assume IX = \$1234, then

LDA \$2A,X ; Loads the content of Page \$12 Locn \$34+\$2A



But IX / IY remain unchanged!!!

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

34



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:

$$(\text{effective address}) = (\text{base address [value in X or Y]}) + (8\text{-bit unsigned offset/displacement})$$
- ❖ 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

35



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 0010₁₆

<p>X \$0010</p> <p>LDAA 0,X STAA DIGIT1 LDAA 1,X STAA DIGIT2 : LDAA 5,X STAA DIGIT6 LDAA 6,X STAA DIGIT7</p>	<p>MEMORY</p> <table border="1" style="margin: auto;"> <tr><td>⋮</td></tr> <tr><td>\$08</td></tr> <tr><td>\$04</td></tr> <tr><td>\$06</td></tr> <tr><td>\$01</td></tr> <tr><td>\$05</td></tr> <tr><td>\$00</td></tr> <tr><td>\$09</td></tr> <tr><td>???</td></tr> <tr><td>⋮</td></tr> </table> <div style="border: 1px solid black; width: 150px; height: 20px; margin: 10px auto;"></div>	⋮	\$08	\$04	\$06	\$01	\$05	\$00	\$09	???	⋮	<p>\$0010 \$0011 \$0012 \$0013 \$0014 \$0015 \$0016 \$0017</p>	<p>Repeat with a better program. This time use a single address for the “Dialer” and use a loop.</p>
⋮													
\$08													
\$04													
\$06													
\$01													
\$05													
\$00													
\$09													
???													
⋮													

phone.asm

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

36



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.

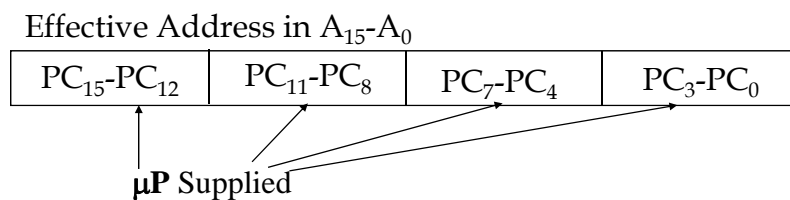
University of Florida, EEL 3701 – File 24
© Drs. Schwartz & Arroyo

37



EEL3701

Immediate Addressing Mode



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.

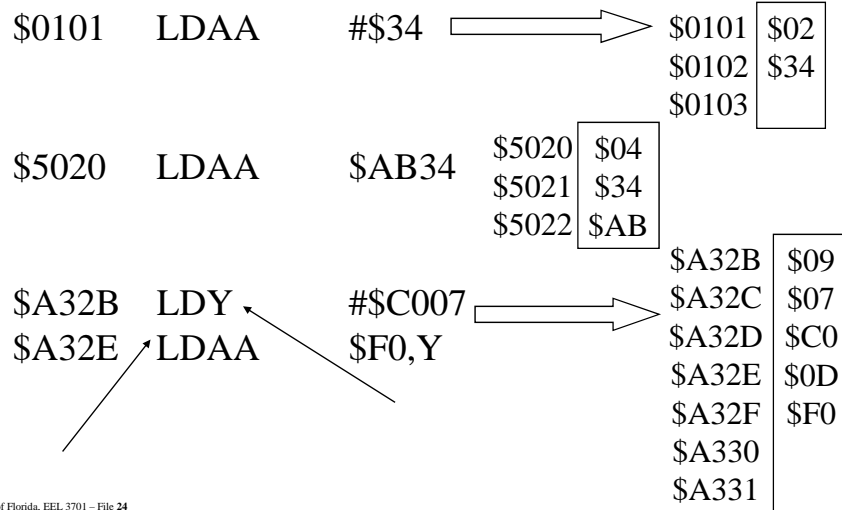
University of Florida, EEL 3701 – File 24
© Drs. Schwartz & Arroyo

38



EEL3701 LDAA Examples with Various Addressing Modes (GCPU)

• Show with memory maps



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

39



EEL3701

Immediate Addressing Mode

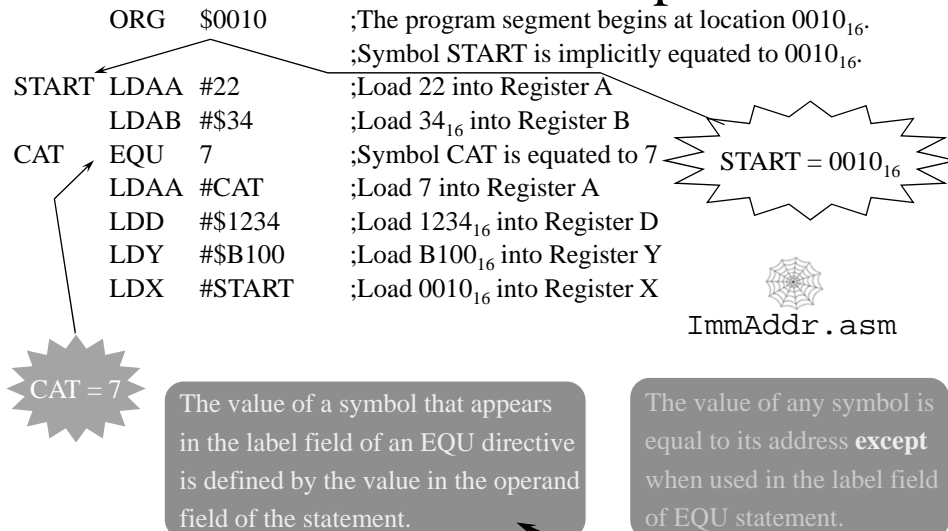
- In the *immediate addressing* mode, the instruction contains the data itself, as an operand. The data can be an 8-bit quantity (a byte), or a 16-bit quantity (a word), depending on the instruction or the destination of the quantity. An immediate operand is indicated by the character # used as a prefix 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

40



EEL3701 Immediate Addressing Examples



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

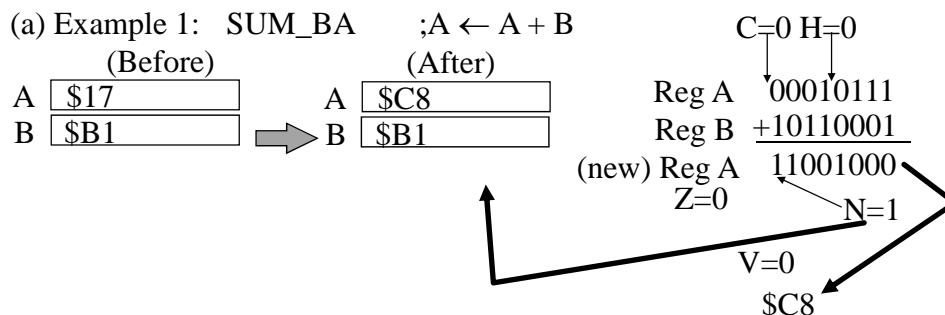
41



EEL3701 Inherent Addressing Example

❖ **Inherent Addressing** Mode improves efficiency

(ex) TAB, SUM_BA



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

42



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!



EEL3701

Common Assembler Directives

- Assembly Control
 - >ORG Origin program counter
- 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



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



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
```



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    $1800
      DC.B   37, $73, 42 ; ($1800) = 37 = $25, ($1801) = $73
*
      DC.B   ; ($1802) = 42 = $2A
GSmrt DC.B   $99          ; (GSmrt) = ($1803) = $99
Table DC.B   3, 9, 44, $2E, 244, $CD ; Table = $1804
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

47



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    $AB42
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

48



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 \$A000 and ending
- * at address \$A0FF. A second table goes from \$A100-\$A1FF. A single 1-byte variable is also shown.

```

      ORG  $A000
Table DS.B 256
Tab2  DS.B 256
Var1  DS.B 1

```

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

49



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 \$A000 and ending
- * at address \$A009. Next is a two-byte variable and then a table of 2 two-byte variables.

```

      ORG  $A000
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

50



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



EEL3701

The End!