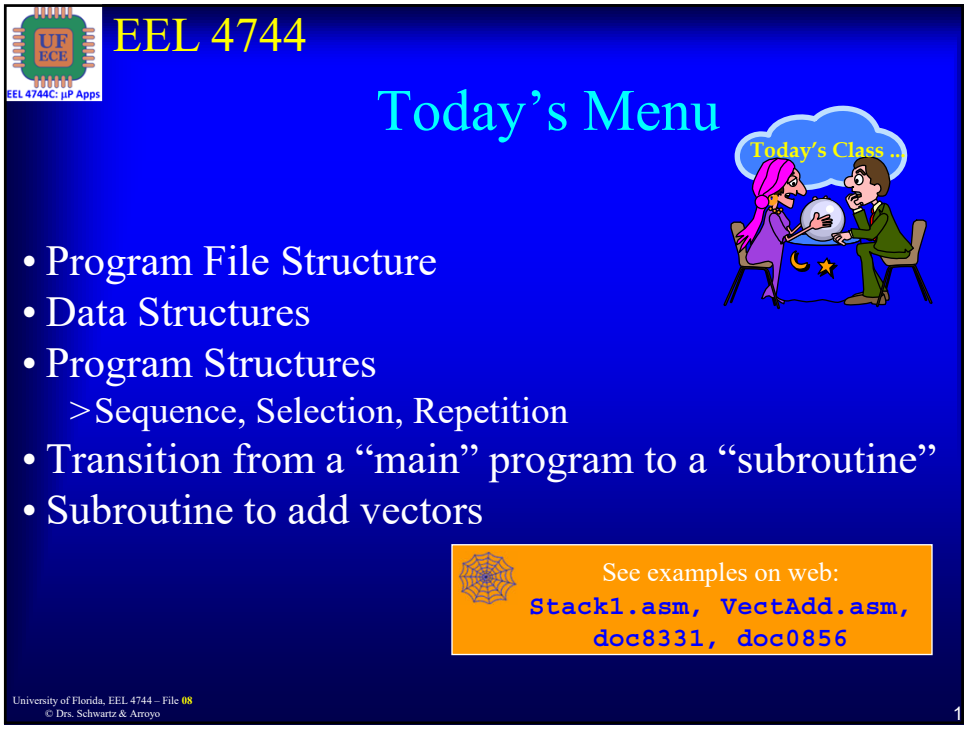
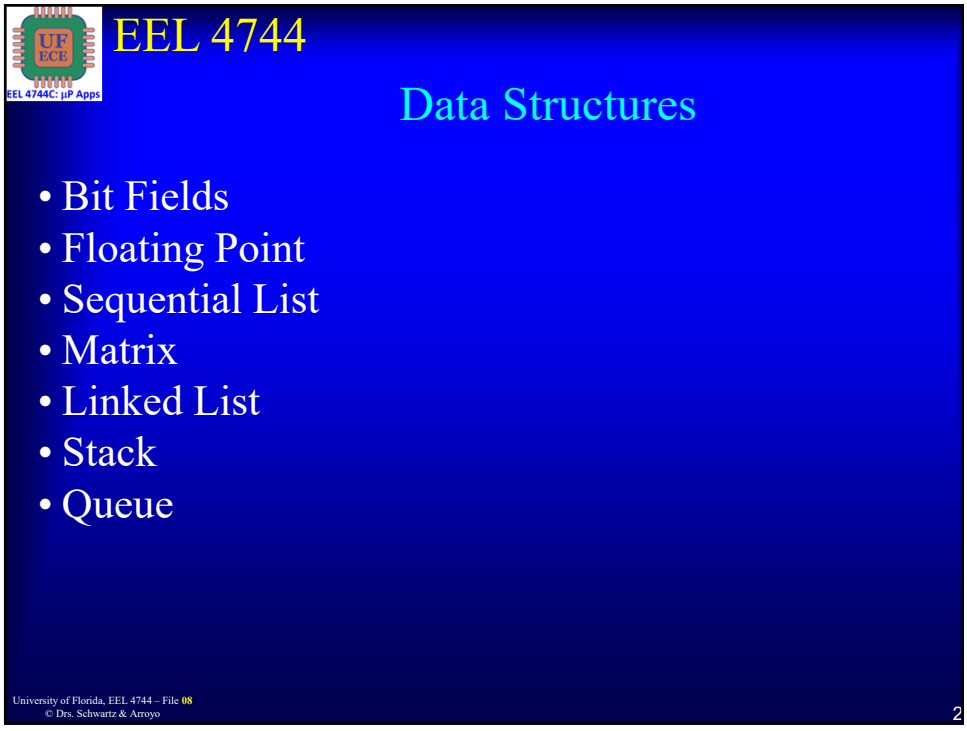

EEL 4744
Today's Menu
Today's Class ..
• Program File Structure
• Data Structures
• Program Structures
>Sequence, Selection, Repetition
• Transition from a "main" program to a "subroutine"
• Subroutine to add vectors
See examples on web:
Stack1.asm, VectAdd.asm,
doc8331, doc0856
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
1

1


EEL 4744
Data Structures
• Bit Fields
• Floating Point
• Sequential List
• Matrix
• Linked List
• Stack
• Queue
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
2

2

EEL 4744

Data Structures: Bit Fields & Floating Point

3

EEL 4744

Data Structures: Sequential List

- A ***sequential list*** of data is a collection of data mapped into successive locations in storage, starting at some initial location called the base address. The order of the elements may or may not have a particular significance.

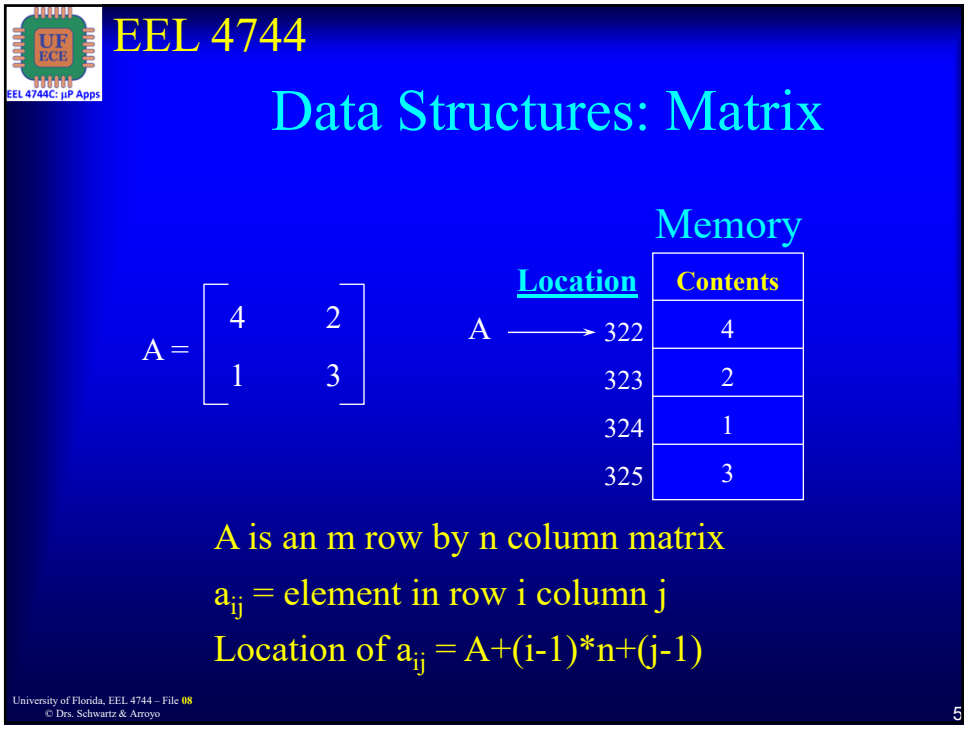
Time	Experimental Data	Location
1	17	122
2	6	123
3	23	124
4	13	125
5	9	126

Contents
17
6
23
13
9

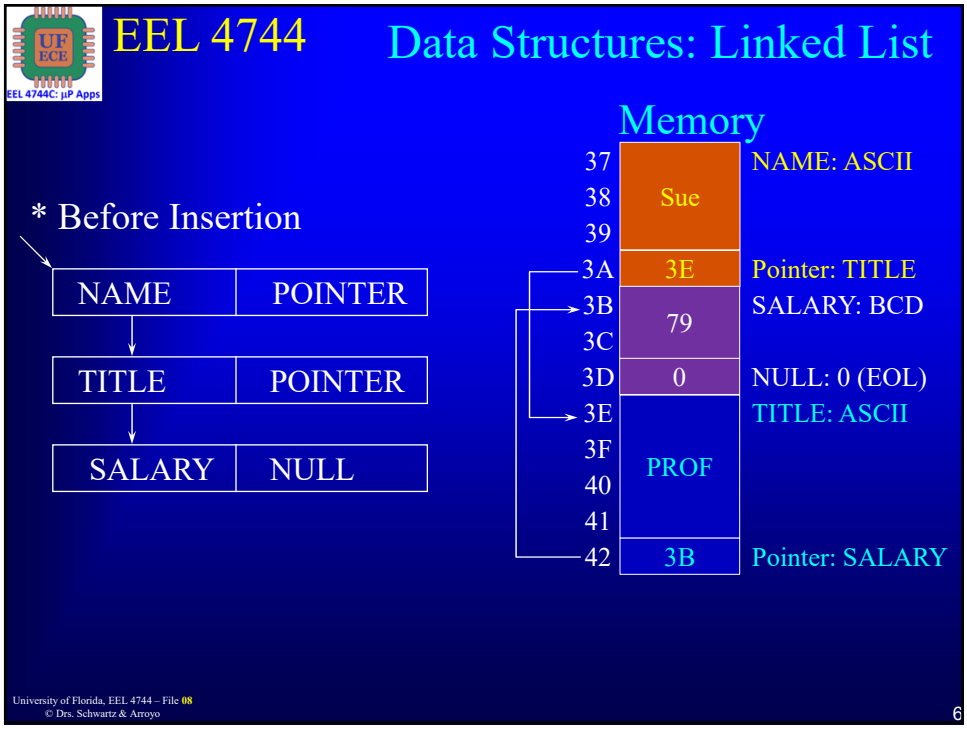
Internal Format Memory

Also called a **TABLE**

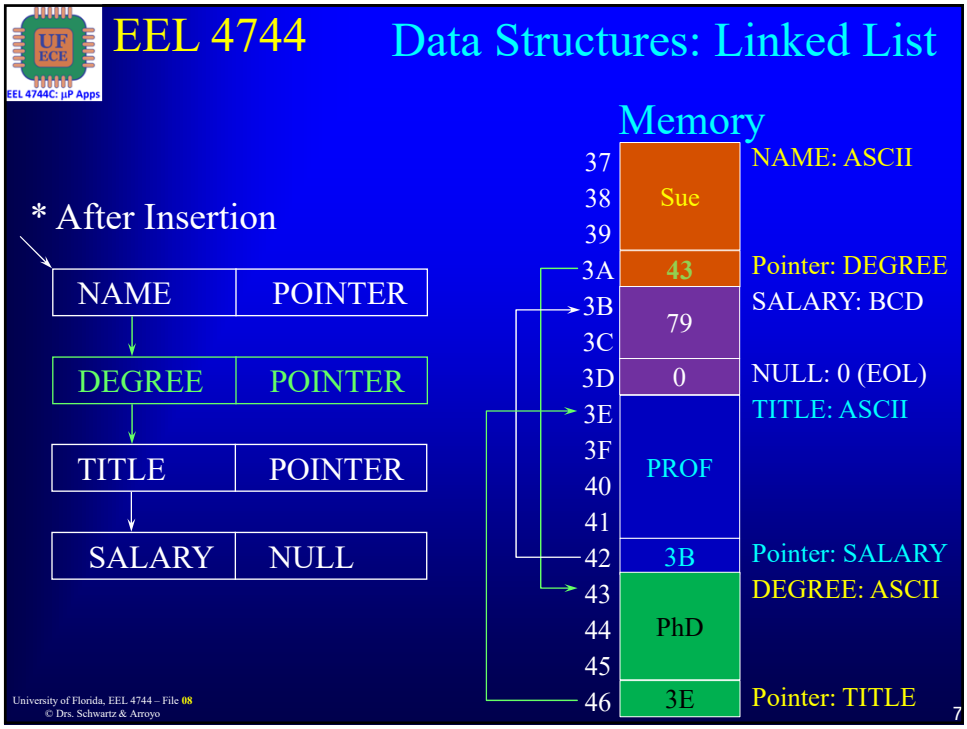
4


EEL 4744
Data Structures: Matrix
Memory
A =
4 2
1 3
A
Location
Contents
322 4
323 2
324 1
325 3
A is an m row by n column matrix
a_{ij} = element in row i column j
Location of a_{ij} = A+(i-1)*n+(j-1)
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
5

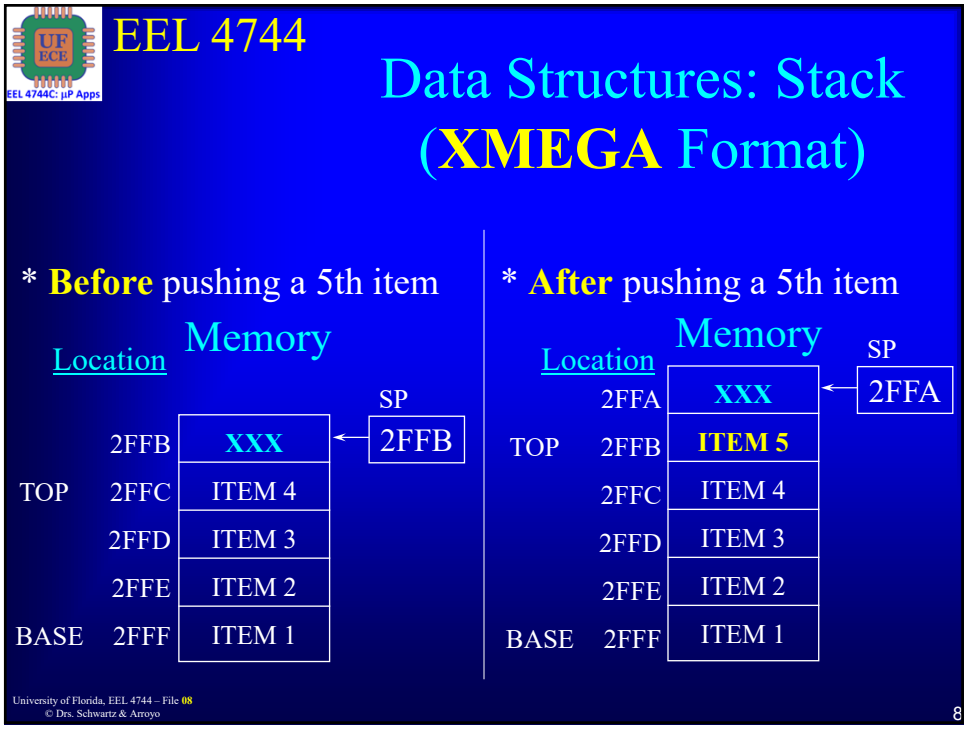
5


EEL 4744
Data Structures: Linked List
Memory
* Before Insertion
NAME POINTER
TITLE POINTER
SALARY NULL
37 Sue NAME: ASCII
38
39
3A 3E Pointer: TITLE
3B 79 SALARY: BCD
3C
3D 0 NULL: 0 (EOL)
3E TITLE: ASCII
3F PROF
40
41
42 3B Pointer: SALARY
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
6

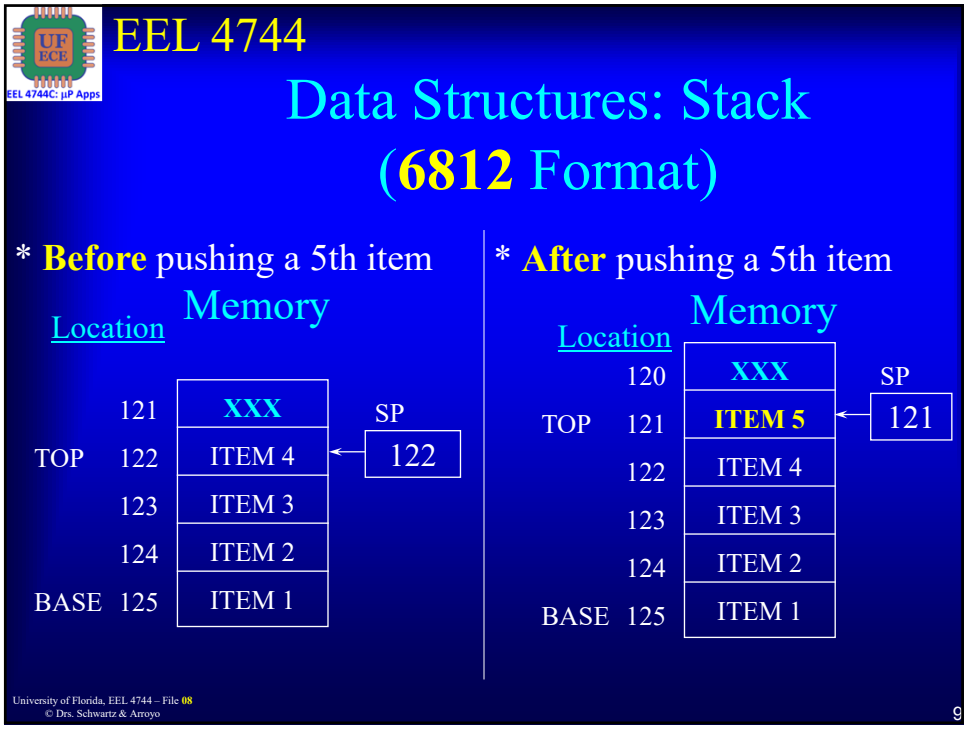
6


EEL 4744
Data Structures: Linked List
EEL 4744C: µP Apps
* After Insertion
NAME
POINTER
DEGREE
POINTER
TITLE
POINTER
SALARY
NULL
Memory
37
38
39
3A
3B
3C
3D
3E
3F
40
41
42
43
44
45
46
Sue
43
79
0
PROF
3B
PhD
3E
NAME: ASCII
Pointer: DEGREE
SALARY: BCD
NULL: 0 (EOL)
TITLE: ASCII
Pointer: SALARY
DEGREE: ASCII
Pointer: TITLE
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
7

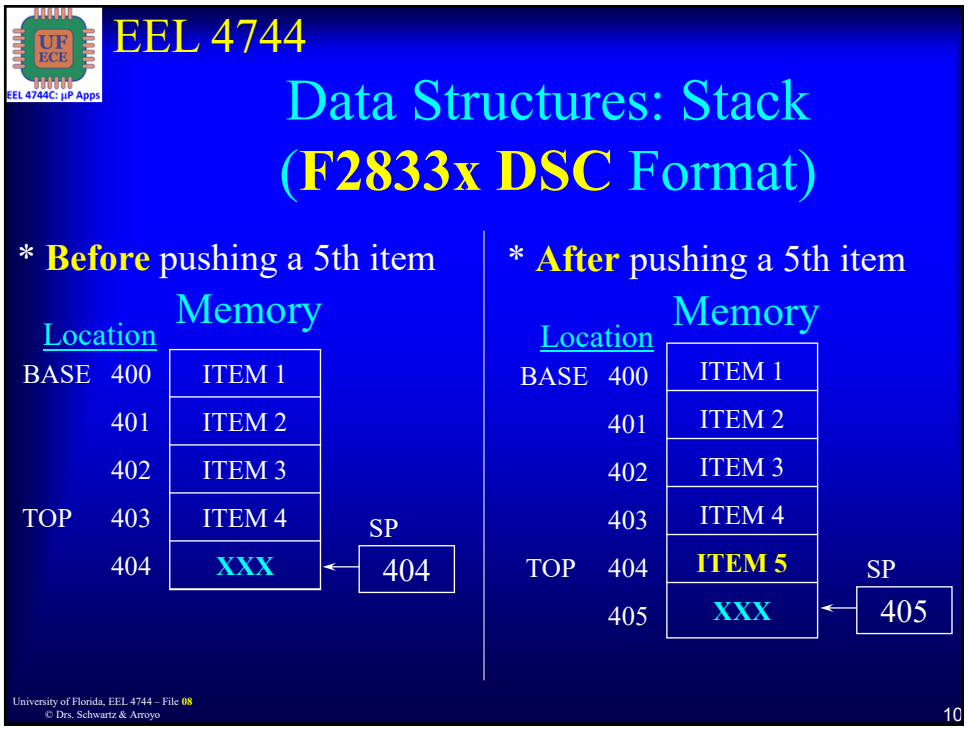
7


EEL 4744
Data Structures: Stack
(XMEGA Format)
EEL 4744C: µP Apps
* Before pushing a 5th item
Location
Memory
SP
2FFB
XXX
2FFB
TOP
2FFC
ITEM 4
2FFD
ITEM 3
2FFE
ITEM 2
BASE
2FFF
ITEM 1
* After pushing a 5th item
Location
Memory
SP
2FFA
XXX
2FFA
TOP
2FFB
ITEM 5
2FFC
ITEM 4
2FFD
ITEM 3
2FFE
ITEM 2
BASE
2FFF
ITEM 1
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
8

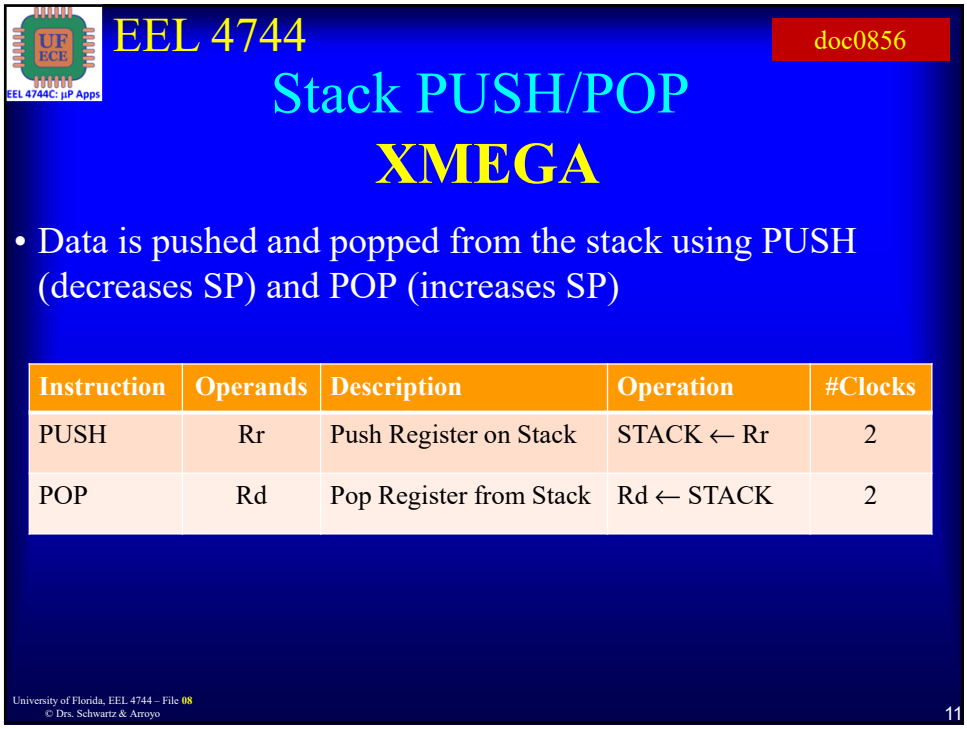
8


EEL 4744
Data Structures: Stack
(6812 Format)
* Before pushing a 5th item
Memory
Location
121 XXX
TOP 122 ITEM 4
123 ITEM 3
124 ITEM 2
BASE 125 ITEM 1
SP
122
* After pushing a 5th item
Memory
Location
120 XXX
TOP 121 ITEM 5
122 ITEM 4
123 ITEM 3
124 ITEM 2
BASE 125 ITEM 1
SP
121
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
9

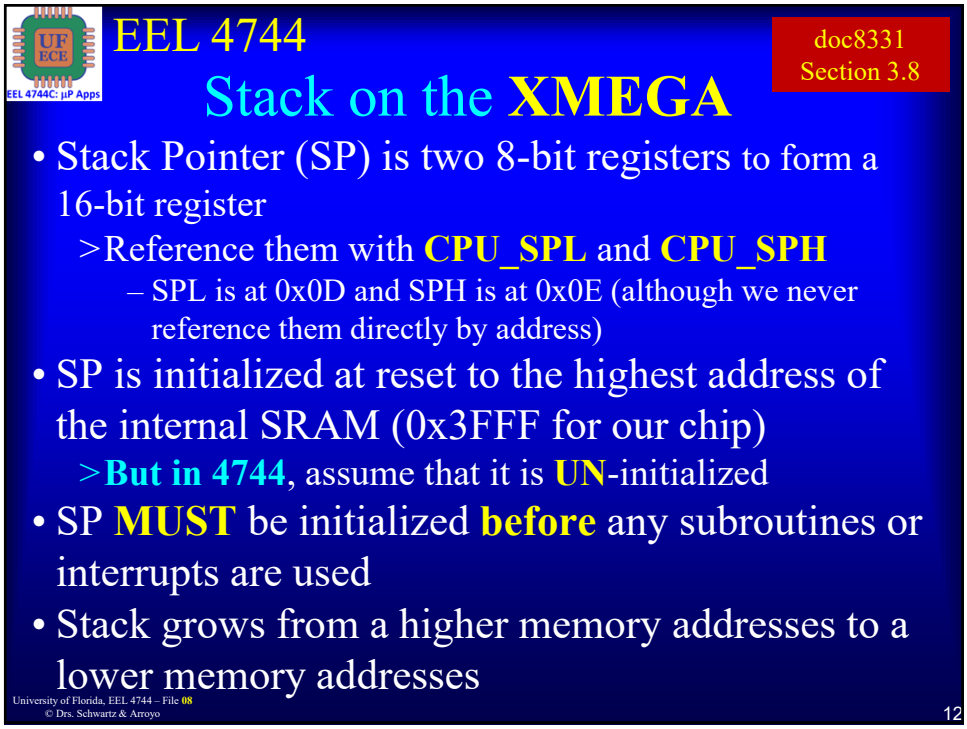
9


EEL 4744
Data Structures: Stack
(F2833x DSC Format)
* Before pushing a 5th item
Memory
Location
BASE 400 ITEM 1
401 ITEM 2
402 ITEM 3
TOP 403 ITEM 4
404 XXX
SP
404
* After pushing a 5th item
Memory
Location
BASE 400 ITEM 1
401 ITEM 2
402 ITEM 3
403 ITEM 4
TOP 404 ITEM 5
405 XXX
SP
405
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
10

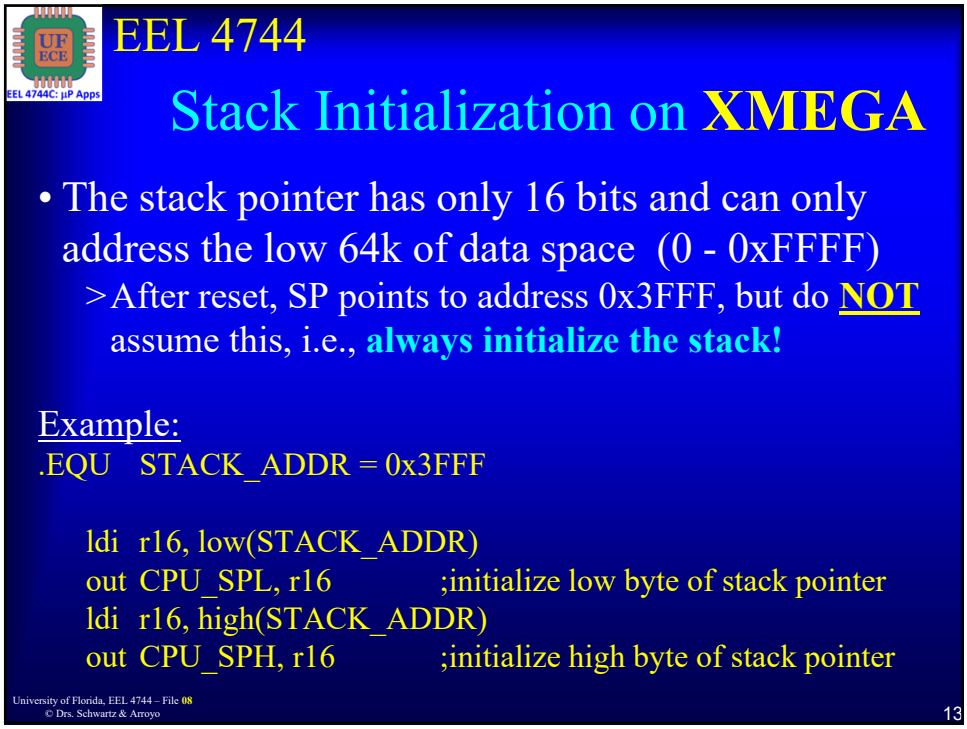
10


EEL 4744
doc0856
Stack PUSH/POP
XMEGA
• Data is pushed and popped from the stack using PUSH (decreases SP) and POP (increases SP)
Instruction | Operands | Description | Operation | #Clocks
PUSH | Rr | Push Register on Stack | STACK ← Rr | 2
POP | Rd | Pop Register from Stack | Rd ← STACK | 2
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
11

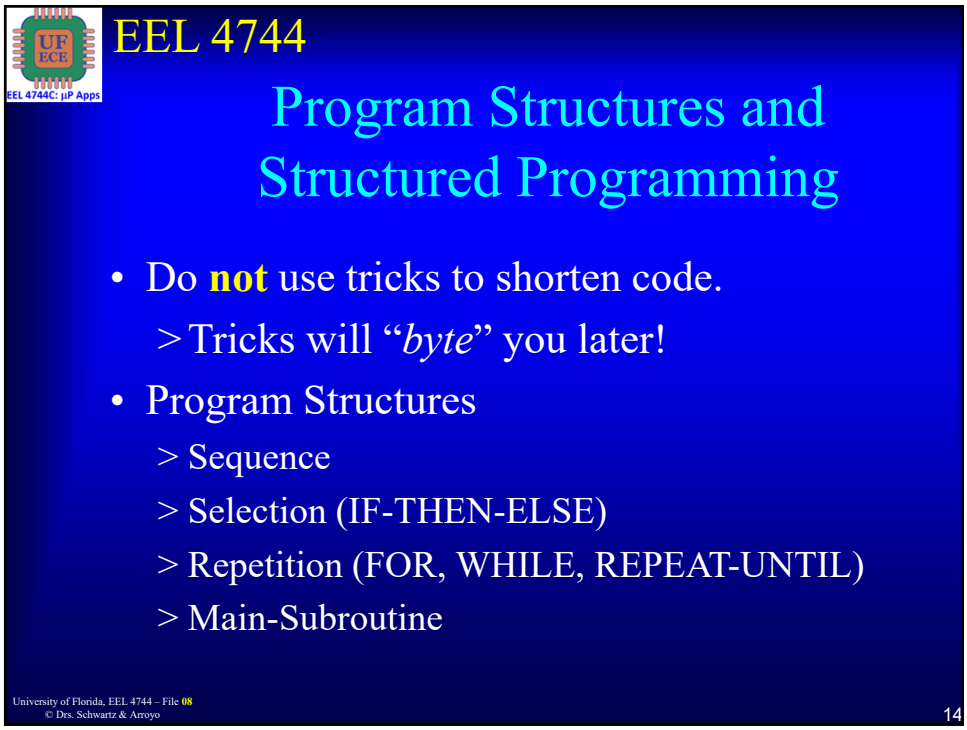
11


EEL 4744
doc8331
Section 3.8
Stack on the XMEGA
• Stack Pointer (SP) is two 8-bit registers to form a 16-bit register
>Reference them with CPU_SPL and CPU_SPH
– SPL is at 0x0D and SPH is at 0x0E (although we never reference them directly by address)
• SP is initialized at reset to the highest address of the internal SRAM (0x3FFF for our chip)
>But in 4744, assume that it is UN-initialized
• SP MUST be initialized before any subroutines or interrupts are used
• Stack grows from a higher memory addresses to a lower memory addresses
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
12

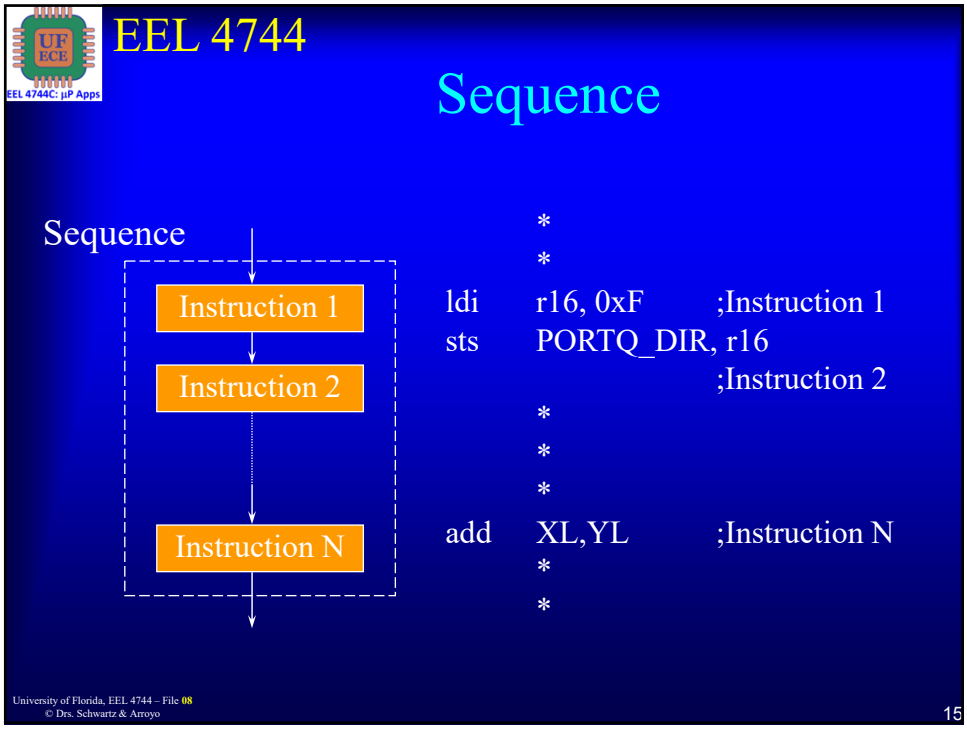
12


EEL 4744
Stack Initialization on XMEGA
• The stack pointer has only 16 bits and can only address the low 64k of data space (0 - 0xFFFF)
> After reset, SP points to address 0x3FFF, but do NOT assume this, i.e., always initialize the stack!
Example:
.EQU STACK_ADDR = 0x3FFF
ldi r16, low(STACK_ADDR)
out CPU_SPL, r16 ;initialize low byte of stack pointer
ldi r16, high(STACK_ADDR)
out CPU_SPH, r16 ;initialize high byte of stack pointer
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
13

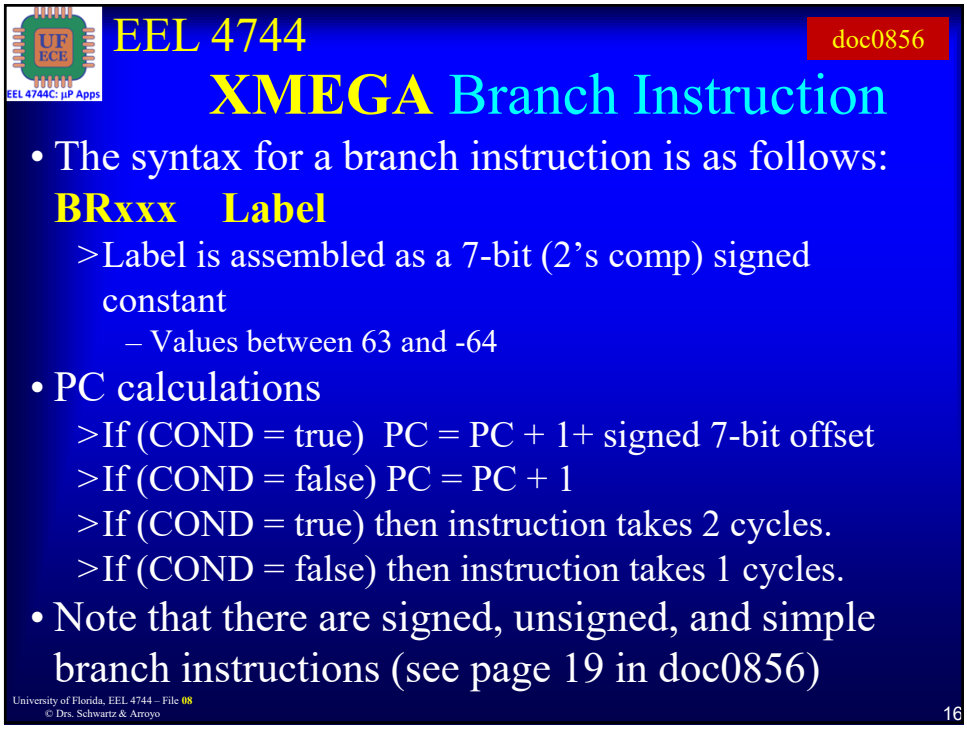
13


EEL 4744
Program Structures and Structured Programming
• Do not use tricks to shorten code.
> Tricks will "byte" you later!
• Program Structures
> Sequence
> Selection (IF-THEN-ELSE)
> Repetition (FOR, WHILE, REPEAT-UNTIL)
> Main-Subroutine
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
14

14


EEL 4744
Sequence
Sequence
Instruction 1
Instruction 2
Instruction N
*
*
ldi r16, 0xF ;Instruction 1
sts PORTQ_DIR, r16
;Instruction 2
*
*
*
add XL,YL ;Instruction N
*
*
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
15

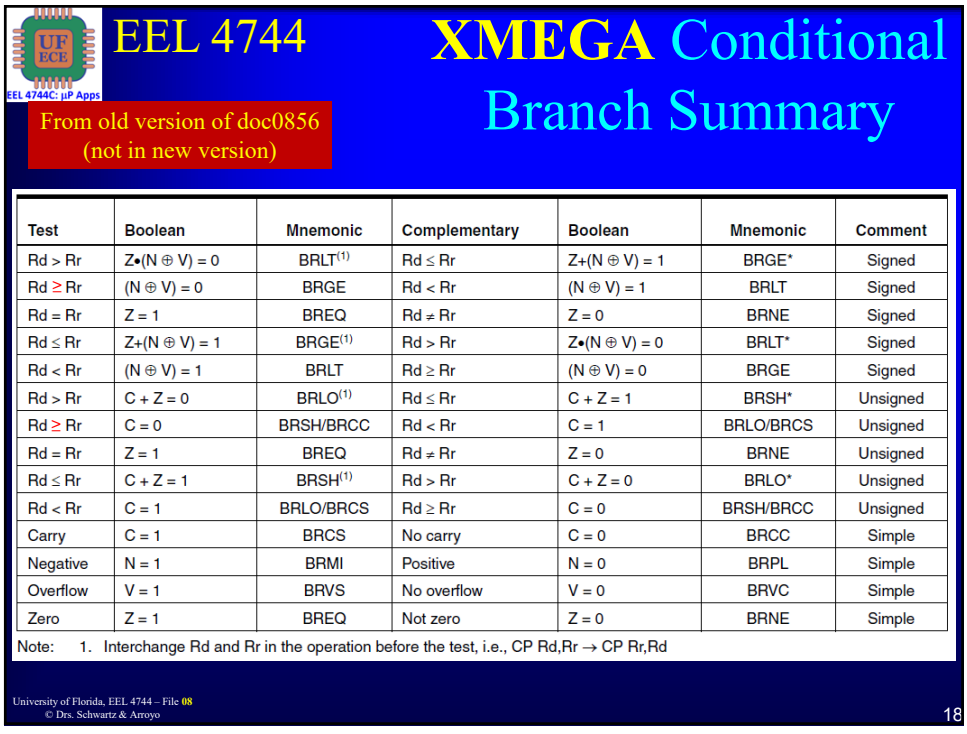
15


EEL 4744
doc0856
XMEGA Branch Instruction
• The syntax for a branch instruction is as follows:
BRxxx Label
>Label is assembled as a 7-bit (2's comp) signed constant
– Values between 63 and -64
• PC calculations
>If (COND = true) PC = PC + 1+ signed 7-bit offset
>If (COND = false) PC = PC + 1
>If (COND = true) then instruction takes 2 cycles.
>If (COND = false) then instruction takes 1 cycles.
• Note that there are signed, unsigned, and simple branch instructions (see page 19 in doc0856)
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
16

16

BRBS	Branch if Status Flag Set	if (SREG(s) = 1) then PC	←	PC + k + 1
BRBC	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC	←	PC + k + 1
BREQ	Branch if Equal	if (Z = 1) then PC	←	PC + k + 1
BRNE	Branch if Not Equal	if (Z = 0) then PC	←	PC + k + 1
BRCS	Branch if Carry Set	if (C = 1) then PC	←	PC + k + 1
BRCC	Branch if Carry Cleared	if (C = 0) then PC	←	PC + k + 1
BRSH	Branch if Same or Higher	if (C = 0) then PC	←	PC + k + 1
BRLO	Branch if Lower	if (C = 1) then PC	←	PC + k + 1
BRMI	Branch if Minus	if (N = 1) then PC	←	PC + k + 1
BRPL	Branch if Plus	if (N = 0) then PC	←	PC + k + 1
BRGE	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC	←	PC + k + 1
BRLT	Branch if Less Than, Signed	if (N ⊕ V= 1) then PC	←	PC + k + 1
BRHS	Branch if Half Carry Flag Set	if (H = 1) then PC	←	PC + k + 1
BRHC	Branch if Half Carry Flag Cleared	if (H = 0) then PC	←	PC + k + 1
BRTS	Branch if T Flag Set	if (T = 1) then PC	←	PC + k + 1
BRTC	Branch if T Flag Cleared	if (T = 0) then PC	←	PC + k + 1
BRVS	Branch if Overflow Flag is Set	if (V = 1) then PC	←	PC + k + 1
BRVC	Branch if Overflow Flag is Cleared	if (V = 0) then PC	←	PC + k + 1
BRIE	Branch if Interrupt Enabled	if (I = 1) then PC	←	PC + k + 1
BRID	Branch if Interrupt Disabled	if (I = 0) then PC	←	PC + k + 1

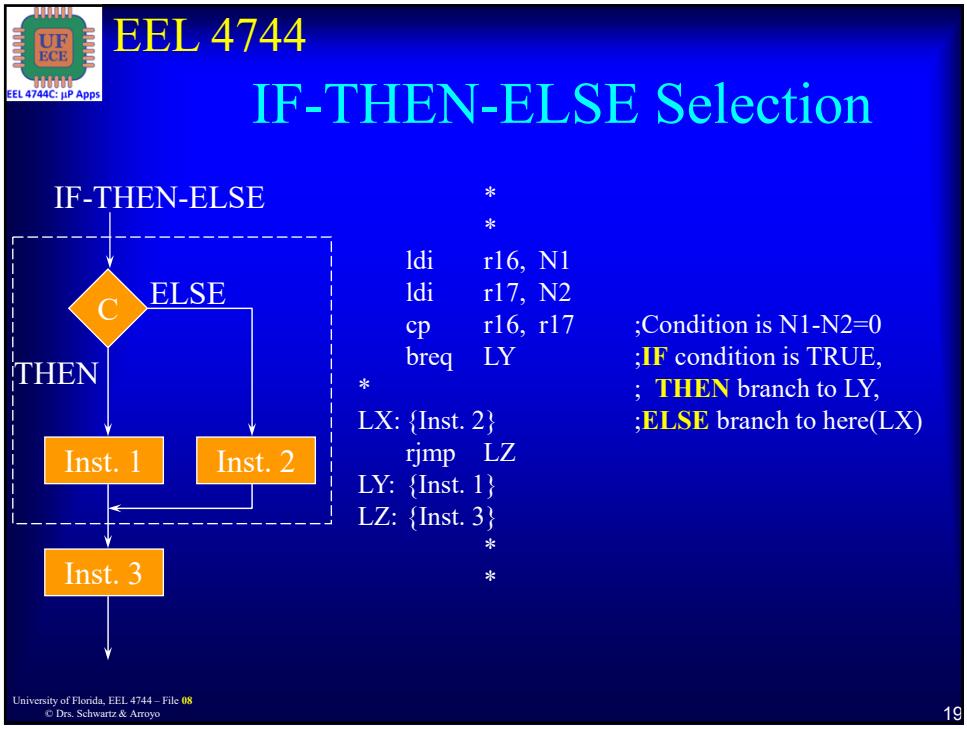
17



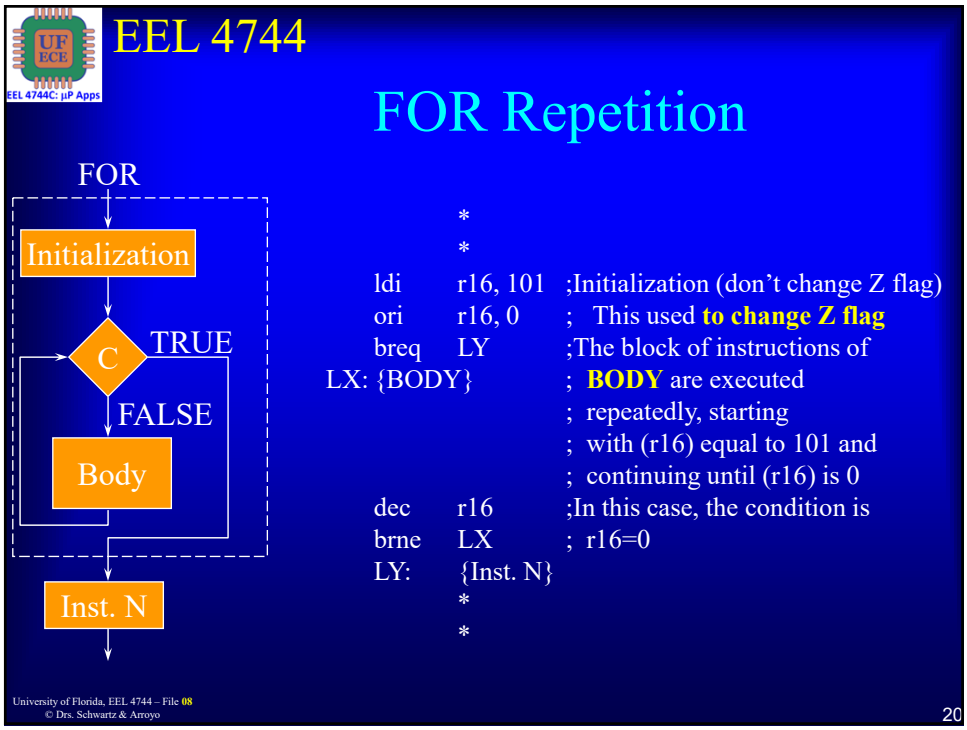
Test	Boolean	Mnemonic	Complementary	Boolean	Mnemonic	Comment
Rd > Rr	Z•(N ⊕ V) = 0	BRLT[1]	Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE*	Signed
Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Rd < Rr	(N ⊕ V) = 1	BRLT	Signed
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Signed
Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE[1]	Rd > Rr	Z•(N ⊕ V) = 0	BRLT*	Signed
Rd < Rr	(N ⊕ V) = 1	BRLT	Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Signed
Rd > Rr	C + Z = 0	BRLO[1]	Rd ≤ Rr	C + Z = 1	BRSH*	Unsigned
Rd ≥ Rr	C = 0	BRSH/BRCC	Rd < Rr	C = 1	BRLO/BRCS	Unsigned
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Unsigned
Rd ≤ Rr	C + Z = 1	BRSH[1]	Rd > Rr	C + Z = 0	BRLO*	Unsigned
Rd < Rr	C = 1	BRLO/BRCS	Rd ≥ Rr	C = 0	BRSH/BRCC	Unsigned
Carry	C = 1	BRCS	No carry	C = 0	BRCC	Simple
Negative	N = 1	BRMI	Positive	N = 0	BRPL	Simple
Overflow	V = 1	BRVS	No overflow	V = 0	BRVC	Simple
Zero	Z = 1	BREQ	Not zero	Z = 0	BRNE	Simple

Note: 1. Interchange Rd and Rr in the operation before the test, i.e., CP Rd,Rr → CP Rr,Rd

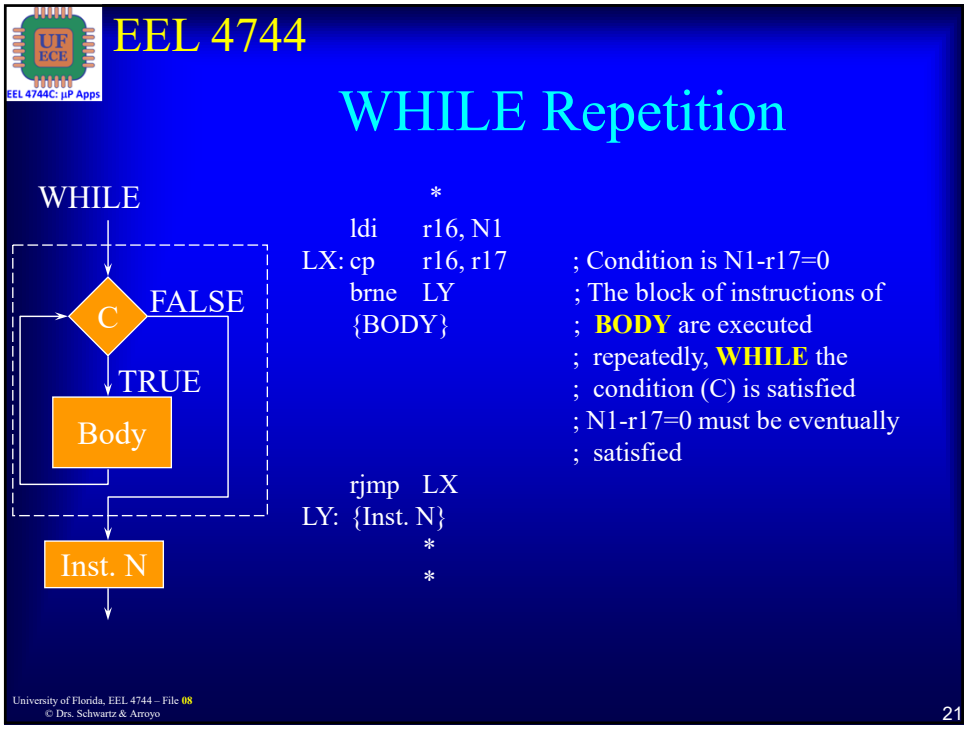
18


EEL 4744
IF-THEN-ELSE Selection
IF-THEN-ELSE
C
ELSE
THEN
Inst. 1
Inst. 2
Inst. 3
*
*
ldi r16, N1
ldi r17, N2
cp r16, r17 ;Condition is N1-N2=0
breq LY ;IF condition is TRUE,
; THEN branch to LY,
* ;ELSE branch to here(LX)
LX: {Inst. 2}
rjmp LZ
LY: {Inst. 1}
LZ: {Inst. 3}
*
*
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
19

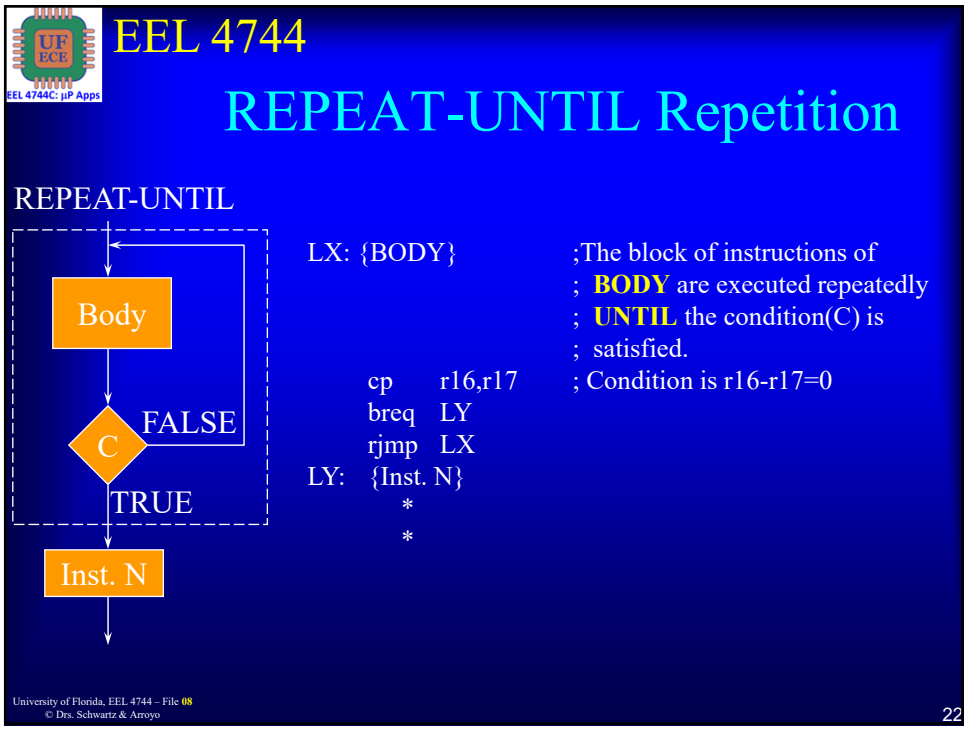
19


EEL 4744
FOR Repetition
FOR
Initialization
C
TRUE
FALSE
Body
Inst. N
*
*
ldi r16, 101 ;Initialization (don't change Z flag)
ori r16, 0 ; This used to change Z flag
breq LY ;The block of instructions of
LX: {BODY} ; BODY are executed
; repeatedly, starting
; with (r16) equal to 101 and
; continuing until (r16) is 0
dec r16 ;In this case, the condition is
brne LX ; r16=0
LY: {Inst. N}
*
*
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
20

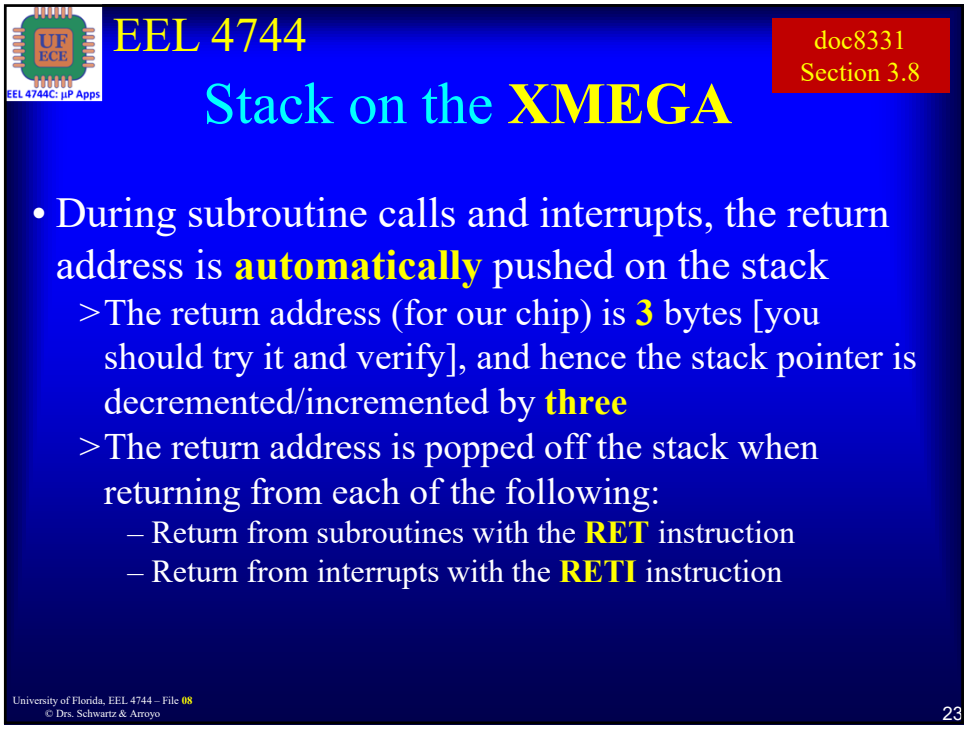
20


EEL 4744
EEL 4744C: µP Apps
WHILE Repetition
WHILE
C
FALSE
TRUE
Body
Inst. N
*
ldi r16, N1
LX: cp r16, r17
brne LY
{BODY}
rjmp LX
LY: {Inst. N}
*
*
; Condition is N1-r17=0
; The block of instructions of
; BODY are executed
; repeatedly, WHILE the
; condition (C) is satisfied
; N1-r17=0 must be eventually
; satisfied
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
21

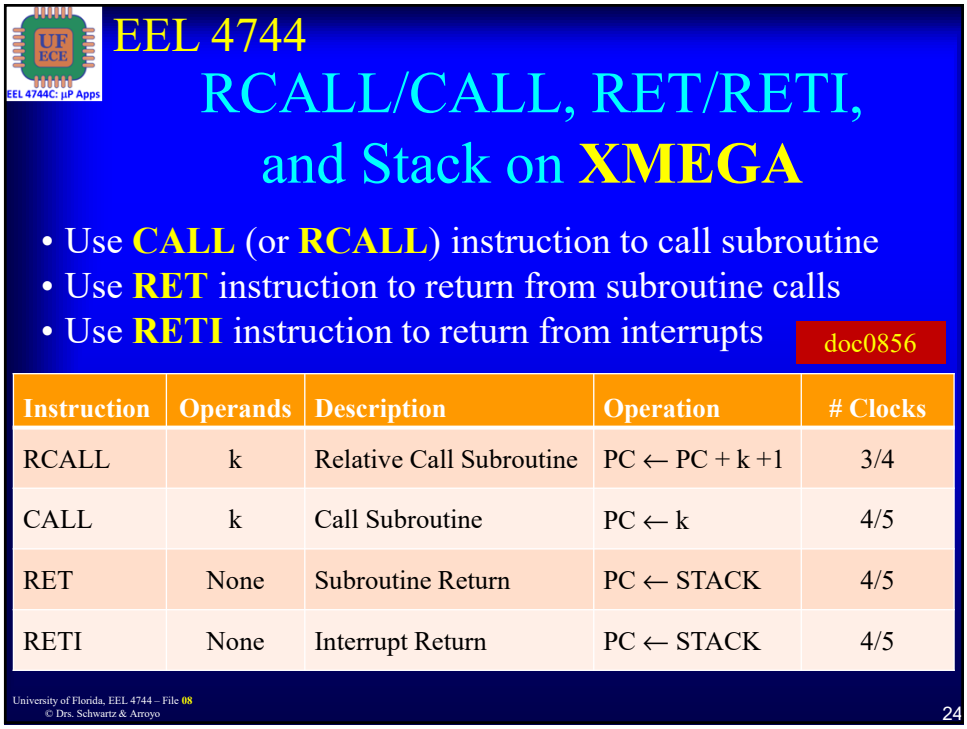
21


EEL 4744
EEL 4744C: µP Apps
REPEAT-UNTIL Repetition
REPEAT-UNTIL
Body
C
FALSE
TRUE
Inst. N
LX: {BODY}
cp r16,r17
breq LY
rjmp LX
LY: {Inst. N}
*
*
;The block of instructions of
; BODY are executed repeatedly
; UNTIL the condition(C) is
; satisfied.
; Condition is r16-r17=0
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
22

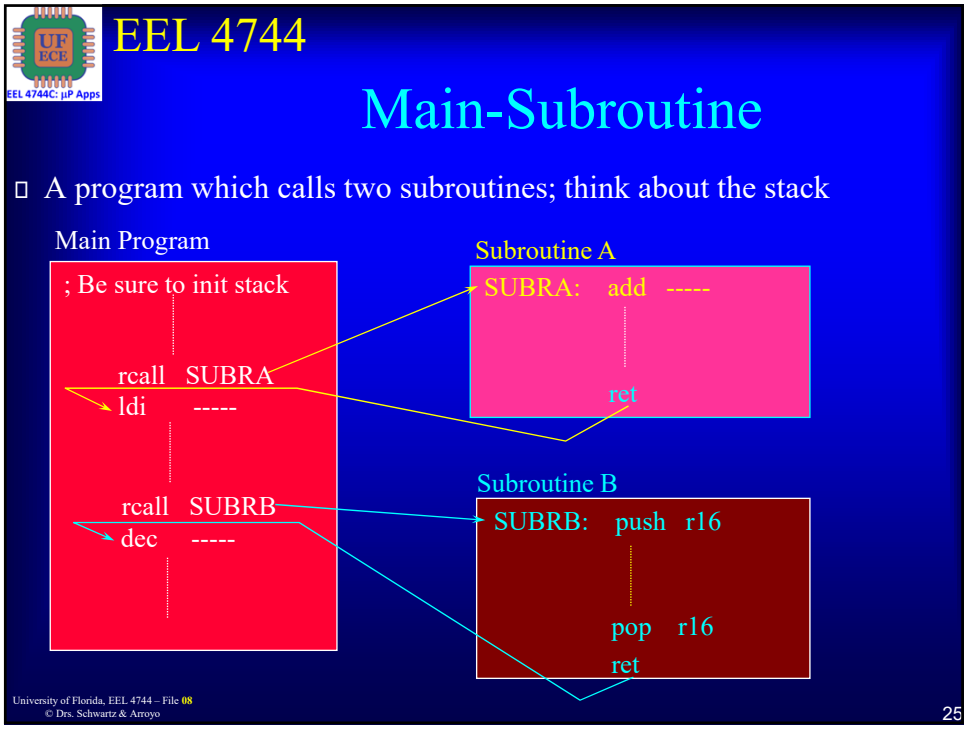
22


EEL 4744
doc8331
Section 3.8
Stack on the XMEGA
• During subroutine calls and interrupts, the return address is automatically pushed on the stack
> The return address (for our chip) is 3 bytes [you should try it and verify], and hence the stack pointer is decremented/incremented by three
> The return address is popped off the stack when returning from each of the following:
– Return from subroutines with the RET instruction
– Return from interrupts with the RETI instruction
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
23

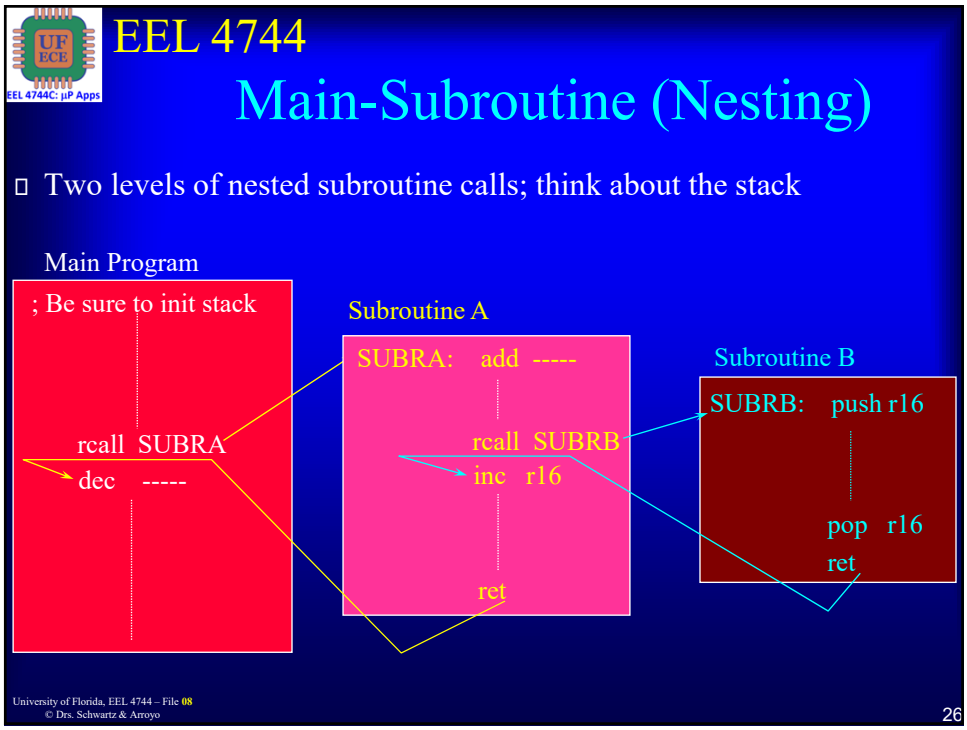
23


EEL 4744
RCALL/CALL, RET/RETI, and Stack on XMEGA
• Use CALL (or RCALL) instruction to call subroutine
• Use RET instruction to return from subroutine calls
• Use RETI instruction to return from interrupts
doc0856
Instruction | Operands | Description | Operation | # Clocks
RCALL | k | Relative Call Subroutine | PC ← PC + k +1 | 3/4
CALL | k | Call Subroutine | PC ← k | 4/5
RET | None | Subroutine Return | PC ← STACK | 4/5
RETI | None | Interrupt Return | PC ← STACK | 4/5
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
24

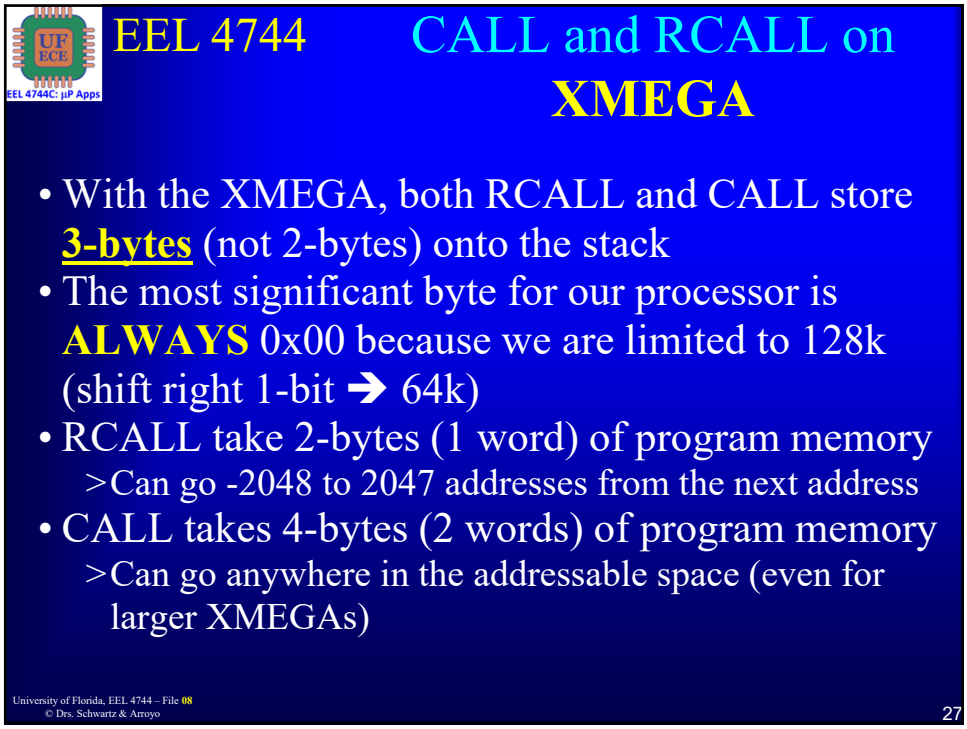
24


EEL 4744
EEL 4744C: µP Apps
Main-Subroutine
A program which calls two subroutines; think about the stack
Main Program
; Be sure to init stack
rcall SUBRA
ldi -----
rcall SUBRB
dec -----
Subroutine A
SUBRA: add -----
ret
Subroutine B
SUBRB: push r16
pop r16
ret
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
25

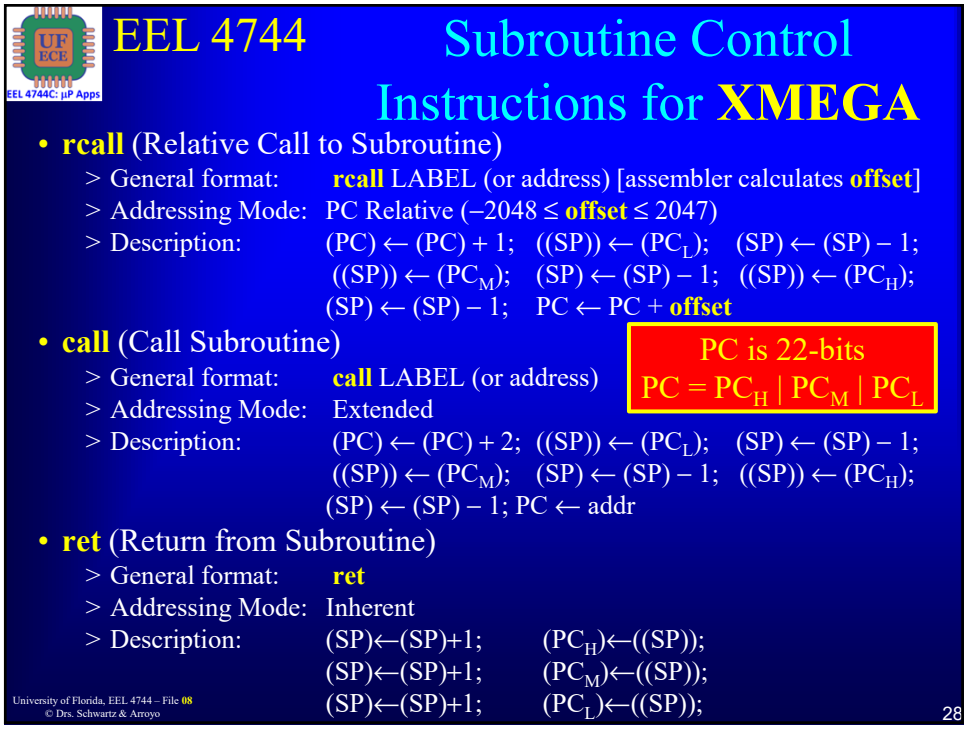
25


EEL 4744
EEL 4744C: µP Apps
Main-Subroutine (Nesting)
Two levels of nested subroutine calls; think about the stack
Main Program
; Be sure to init stack
rcall SUBRA
dec -----
Subroutine A
SUBRA: add -----
rcall SUBRB
inc r16
ret
Subroutine B
SUBRB: push r16
pop r16
ret
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
26

26


EEL 4744
CALL and RCALL on XMEGA
• With the XMEGA, both RCALL and CALL store 3-bytes (not 2-bytes) onto the stack
• The most significant byte for our processor is ALWAYS 0x00 because we are limited to 128k (shift right 1-bit ➔ 64k)
• RCALL take 2-bytes (1 word) of program memory
>Can go -2048 to 2047 addresses from the next address
• CALL takes 4-bytes (2 words) of program memory
>Can go anywhere in the addressable space (even for larger XMEGAs)
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
27

27


EEL 4744
Subroutine Control Instructions for XMEGA
• rcall (Relative Call to Subroutine)
> General format: rcall LABEL (or address) [assembler calculates offset]
> Addressing Mode: PC Relative (–2048 ≤ offset ≤ 2047)
> Description: (PC) ← (PC) + 1; ((SP)) ← (PCL); (SP) ← (SP) – 1; ((SP)) ← (PCM); (SP) ← (SP) – 1; ((SP)) ← (PCH); (SP) ← (SP) – 1; PC ← PC + offset
• call (Call Subroutine)
PC is 22-bits
PC = PCH | PCM | PCL
> General format: call LABEL (or address)
> Addressing Mode: Extended
> Description: (PC) ← (PC) + 2; ((SP)) ← (PCL); (SP) ← (SP) – 1; ((SP)) ← (PCM); (SP) ← (SP) – 1; ((SP)) ← (PCH); (SP) ← (SP) – 1; PC ← addr
• ret (Return from Subroutine)
> General format: ret
> Addressing Mode: Inherent
> Description: (SP)←(SP)+1; (PCH)←((SP)); (SP)←(SP)+1; (PCM)←((SP)); (SP)←(SP)+1; (PCL)←((SP));
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
28

28

EEL 4744

XMEGA Stack Example with Subroutine

- See example on website: **Stack1.asm**
  - >View code and **simulate**
    - – Watch stack, stack pointer (SP)

Stack1.asm

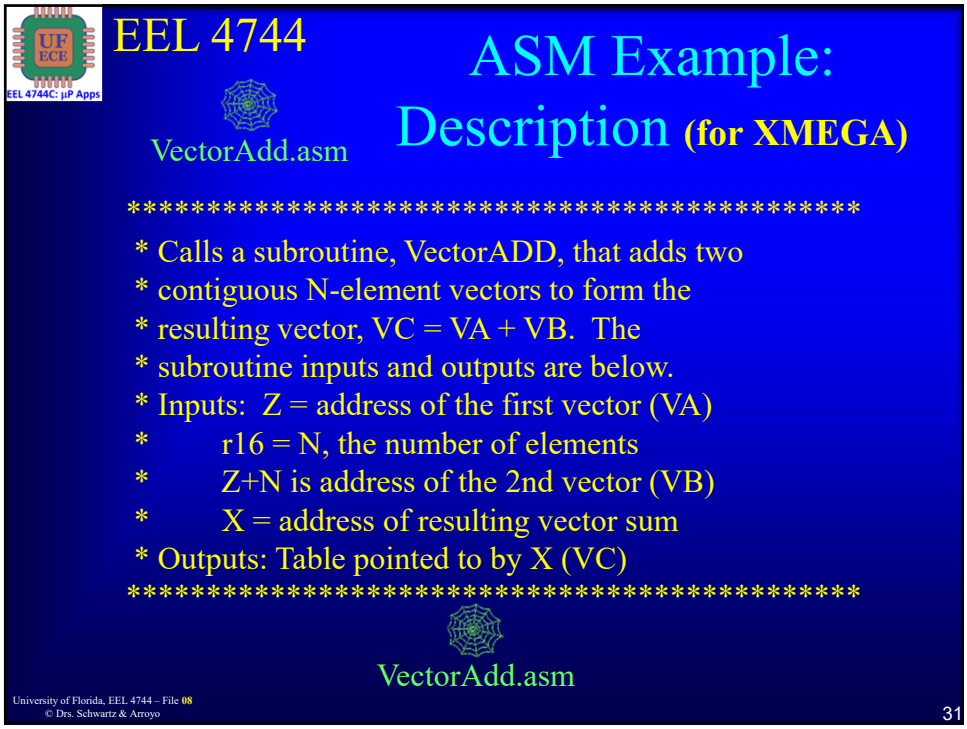
29

EEL 4744

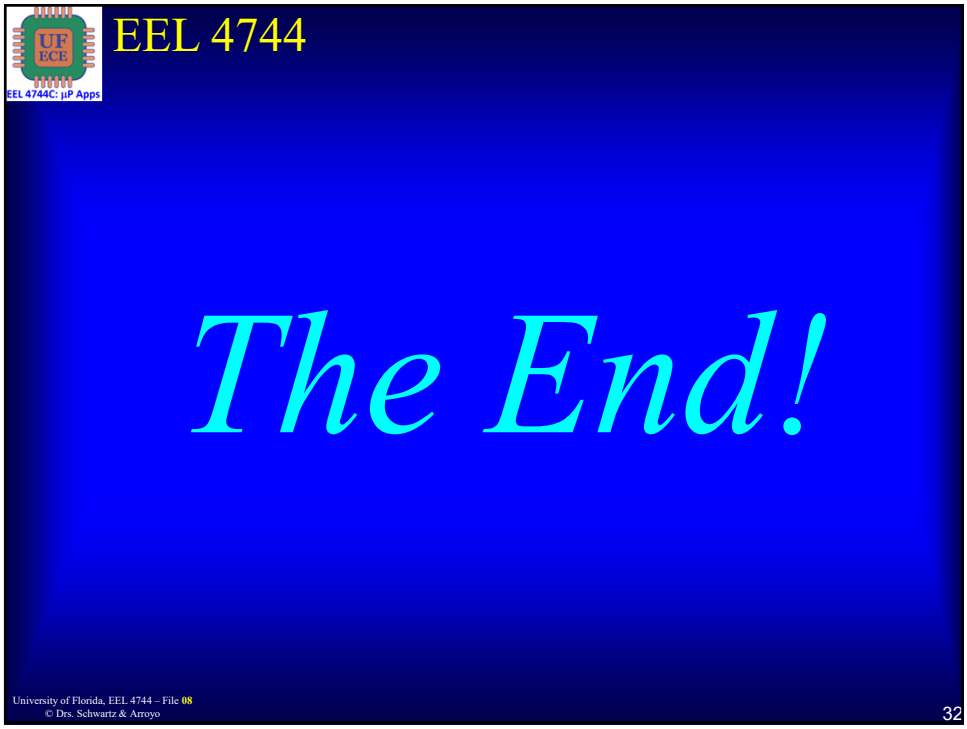
Example (Add two Vectors)

```
void Add_Vectors(int a[3], int b[3], int result[3]);
int main(void)
{
   int A[3] = {1, 2, 3};
   int B[3] = {0xA0, 0xB0, 0xC0};
   int C[3];
   Add_Vectors(A, B, C);
}
void Add_Vectors(int a[3], int b[3], int result[3])
{
    int i;
    for(i=0; i<3; i++)
    {
        result[i] = a[i] + b[i];
    }
}
```


30


EEL 4744
EEL 4744C: µP Apps
ASM Example:
Description (for XMEGA)
VectorAdd.asm
****
* Calls a subroutine, VectorADD, that adds two
* contiguous N-element vectors to form the
* resulting vector, VC = VA + VB. The
* subroutine inputs and outputs are below.
* Inputs: Z = address of the first vector (VA)
* r16 = N, the number of elements
* Z+N is address of the 2nd vector (VB)
* X = address of resulting vector sum
* Outputs: Table pointed to by X (VC)
****
VectorAdd.asm
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
31

31


EEL 4744
EEL 4744C: µP Apps
The End!
University of Florida, EEL 4744 – File 08
© Drs. Schwartz & Arroyo
32

32