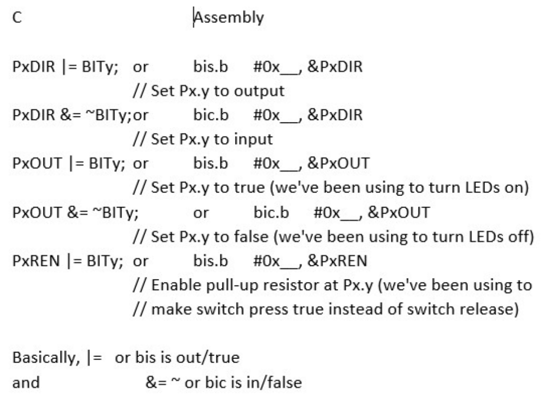
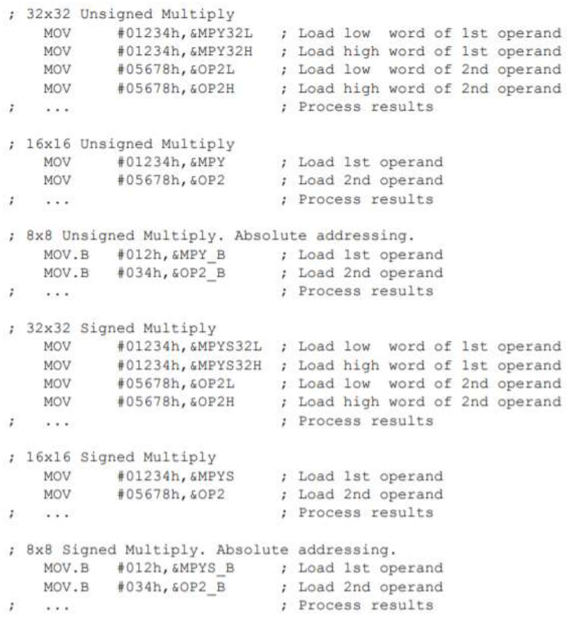
CPE 325 study guide

- Stack
  - The stack is a last-in-first-out structure. When something is moved onto the stack using a PUSH.W instruction, the SP decrements by 2 (lowers / points to the next lowest word in memory) and then stores the value there, so the stack grows toward lower addresses. In the MSP430 that we've been using for labs, the stack starts at address 0x4400. When the first value is pushed, the SP points at address 0x43FE and the value is stored there. Using POP.W takes the value from the address that the SP is currently pointing to, stores it in the destination specified by the pop instruction, and increments the SP, so the stack shrinks toward higher addresses.
- Ports



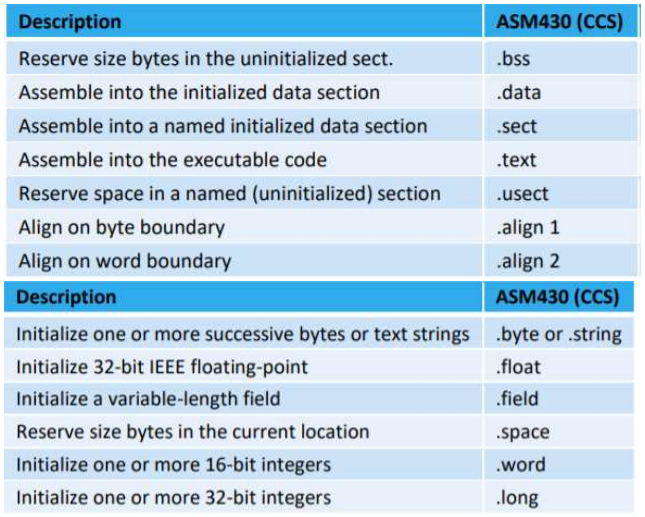
```
C                       Assembly

PxDIR |= BITy;   or     bis.b    #0x__, &PxDIR
                 // Set Px.y to output
PxDIR &= ~BITy;or       bic.b    #0x__, &PxDIR
                 // Set Px.y to input
PxOUT |= BITy;  or      bis.b    #0x__, &PxOUT
                 // Set Px.y to true (we've been using to turn LEDs on)
PxOUT &= ~BITy;         or       bic.b    #0x__, &PxOUT
                 // Set Px.y to false (we've been using to turn LEDs off)
PxREN |= BITy;  or      bis.b    #0x__, &PxREN
                 // Enable pull-up resistor at Px.y (we've been using to
                 // make switch press true instead of switch release)

Basically, |=  or bis is out/true
and            &= ~ or bic is in/false
```

- Hardware Multiplier



```
; 32x32 Unsigned Multiply
    MOV     #01234h,&MPY32L   ; Load low  word of 1st operand
    MOV     #01234h,&MPY32H   ; Load high word of 1st operand
    MOV     #05678h,&OP2L     ; Load low  word of 2nd operand
    MOV     #05678h,&OP2H     ; Load high word of 2nd operand
;   ...                       ; Process results

; 16x16 Unsigned Multiply
    MOV     #01234h,&MPY      ; Load 1st operand
    MOV     #05678h,&OP2      ; Load 2nd operand
;   ...                       ; Process results

; 8x8 Unsigned Multiply. Absolute addressing.
    MOV.B   #012h,&MPY_B      ; Load 1st operand
    MOV.B   #034h,&OP2_B      ; Load 2nd operand
;   ...                       ; Process results

; 32x32 Signed Multiply
    MOV     #01234h,&MPYS32L  ; Load low  word of 1st operand
    MOV     #01234h,&MPYS32H  ; Load high word of 1st operand
    MOV     #05678h,&OP2L     ; Load low  word of 2nd operand
    MOV     #05678h,&OP2H     ; Load high word of 2nd operand
;   ...                       ; Process results

; 16x16 Signed Multiply
    MOV     #01234h,&MPYS     ; Load 1st operand
    MOV     #05678h,&OP2      ; Load 2nd operand
;   ...                       ; Process results

; 8x8 Signed Multiply. Absolute addressing.
    MOV.B   #012h,&MPYS_B     ; Load 1st operand
    MOV.B   #034h,&OP2_B      ; Load 2nd operand
;   ...                       ; Process results
```

- Memory Allocation
  - .bss and .data start at 0x2400 for the MSP430 from the labs



Description	ASM430 (CCS)
Reserve size bytes in the uninitialized sect.	.bss
Assemble into the initialized data section	.data
Assemble into a named initialized data section	.sect
Assemble into the executable code	.text
Reserve space in a named (uninitialized) section	.usect
Align on byte boundary	.align 1
Align on word boundary	.align 2

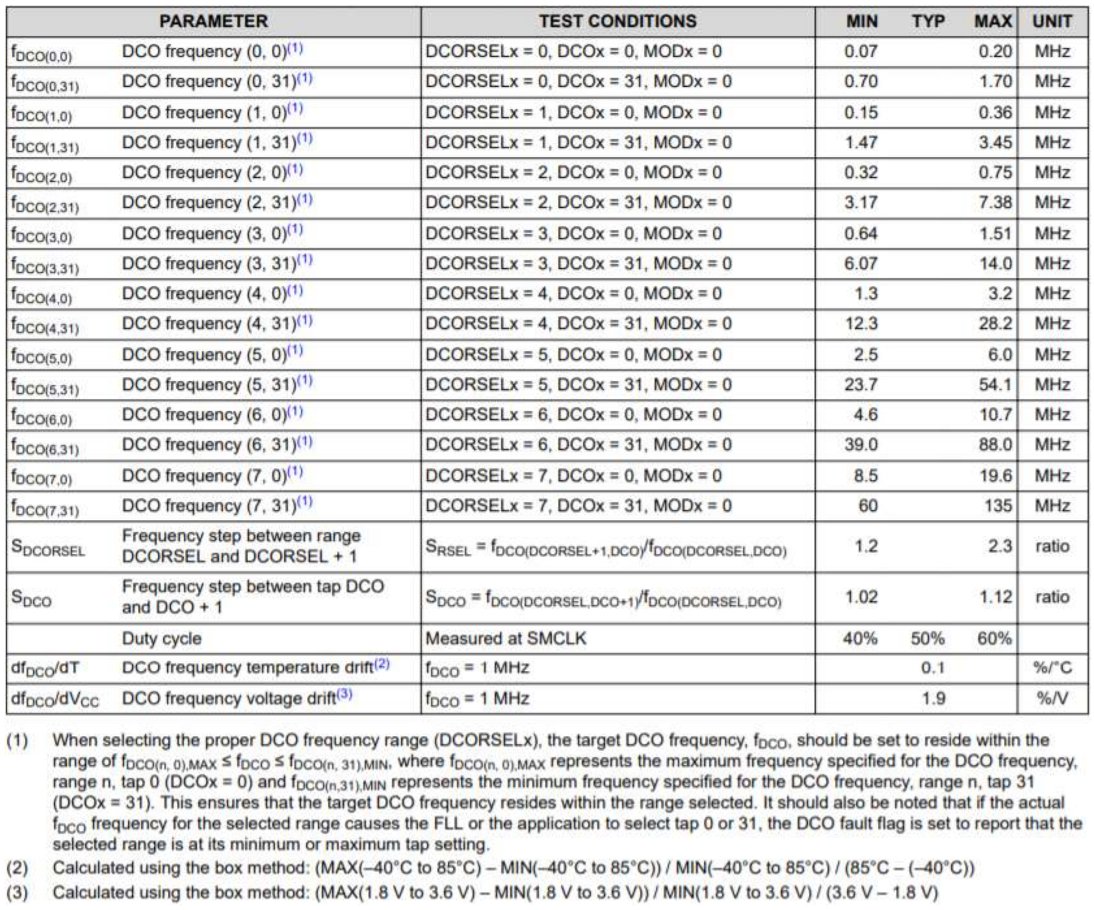
Description	ASM430 (CCS)
Initialize one or more successive bytes or text strings	.byte or .string
Initialize 32-bit IEEE floating-point	.float
Initialize a variable-length field	.field
Reserve size bytes in the current location	.space
Initialize one or more 16-bit integers	.word
Initialize one or more 32-bit integers	.long

- Subroutines
  - These are like C's user-defined functions for assembly. Execution is moved to the subroutine using the CALL instruction. What this does is push the current value of the PC onto the stack (the return address) and then changes the PC to be at the subroutine. At the end of a subroutine, use the RET instruction to go back to where it was called from. This basically acts like POP PC. It pops whatever is at the bottom of the stack and puts that in the PC. It's important to keep track of this and make sure that the SP is pointing to the correct return address at the end of a subroutine. For every PUSH instruction in a subroutine, there needs to be a POP instruction.
- Parallel Ports
  - This basically means that a single port can operate on both input and output as far as I'm aware
- Control Registers
  - These are special registers in the processor whose values are linked to different hardware functions. For example, see the port mapping section hardware multiplier section
- Interrupts
  - Basically a subroutine. It can be helpful to think of it like a moving CALL instruction that's always 1 line ahead of the PC. Whenever the interrupt is activated, the CALL stops and allows the PC to execute it. So the interrupt can be activated from anywhere in the program. However, multiple interrupts can't activate until the current one has returned to main.

```
C                           Assembly

_EINT();          or        bis.w    #GIE, SR
                  // Enable global interrupts
PxIE |= BITy;     or        bis.b    #0x__, &PxIE
                  // Enable interrupts at Px.y
PxIES |= BITy;    or        bis.b    #0x__, &PxIES
                  // Switch button press activates interrupt
PxIFG &= ~BITy;   or        bic.b    #0x__, &PxIFG
                  // Clear interrupt flag for Px.y
```

```
.sect    ".int47"     ; Port 1 vector
.short   SW2_ISR
.sect    ".int42"     ; Port 2 vector
.short   SW1_ISR
```

- Clock stuff
    - IDK. There's a lot going on here and I don't feel like trying to explain it. I know it well enough to do the lab and maybe recreate some stuff for the quiz, but not enough to explain it. Here's a chart tho.



PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$f_{DCO(0,0)}$	DCO frequency (0, 0)(1)	DCORSELx = 0, DCOx = 0, MODx = 0	0.07		0.20	MHz
$f_{DCO(0,31)}$	DCO frequency (0, 31)(1)	DCORSELx = 0, DCOx = 31, MODx = 0	0.70		1.70	MHz
$f_{DCO(1,0)}$	DCO frequency (1, 0)(1)	DCORSELx = 1, DCOx = 0, MODx = 0	0.15		0.36	MHz
$f_{DCO(1,31)}$	DCO frequency (1, 31)(1)	DCORSELx = 1, DCOx = 31, MODx = 0	1.47		3.45	MHz
$f_{DCO(2,0)}$	DCO frequency (2, 0)(1)	DCORSELx = 2, DCOx = 0, MODx = 0	0.32		0.75	MHz
$f_{DCO(2,31)}$	DCO frequency (2, 31)(1)	DCORSELx = 2, DCOx = 31, MODx = 0	3.17		7.38	MHz
$f_{DCO(3,0)}$	DCO frequency (3, 0)(1)	DCORSELx = 3, DCOx = 0, MODx = 0	0.64		1.51	MHz
$f_{DCO(3,31)}$	DCO frequency (3, 31)(1)	DCORSELx = 3, DCOx = 31, MODx = 0	6.07		14.0	MHz
$f_{DCO(4,0)}$	DCO frequency (4, 0)(1)	DCORSELx = 4, DCOx = 0, MODx = 0	1.3		3.2	MHz
$f_{DCO(4,31)}$	DCO frequency (4, 31)(1)	DCORSELx = 4, DCOx = 31, MODx = 0	12.3		28.2	MHz
$f_{DCO(5,0)}$	DCO frequency (5, 0)(1)	DCORSELx = 5, DCOx = 0, MODx = 0	2.5		6.0	MHz
$f_{DCO(5,31)}$	DCO frequency (5, 31)(1)	DCORSELx = 5, DCOx = 31, MODx = 0	23.7		54.1	MHz
$f_{DCO(6,0)}$	DCO frequency (6, 0)(1)	DCORSELx = 6, DCOx = 0, MODx = 0	4.6		10.7	MHz
$f_{DCO(6,31)}$	DCO frequency (6, 31)(1)	DCORSELx = 6, DCOx = 31, MODx = 0	39.0		88.0	MHz
$f_{DCO(7,0)}$	DCO frequency (7, 0)(1)	DCORSELx = 7, DCOx = 0, MODx = 0	8.5		19.6	MHz
$f_{DCO(7,31)}$	DCO frequency (7, 31)(1)	DCORSELx = 7, DCOx = 31, MODx = 0	60		135	MHz
$S_{DCORSEL}$	Frequency step between range DCORSEL and DCORSEL + 1	$S_{RSEL} = f_{DCO(DCORSEL+1,DCO)}/f_{DCO(DCORSEL,DCO)}$	1.2		2.3	ratio
S_{DCO}	Frequency step between tap DCO and DCO + 1	$S_{DCO} = f_{DCO(DCORSEL,DCO+1)}/f_{DCO(DCORSEL,DCO)}$	1.02		1.12	ratio
	Duty cycle	Measured at SMCLK	40%	50%	60%	
df_{DCO}/dT	DCO frequency temperature drift(2)	f_{DCO} = 1 MHz		0.1		%/°C
df_{DCO}/dV_{CC}	DCO frequency voltage drift(3)	f_{DCO} = 1 MHz		1.9		%/V

(1) When selecting the proper DCO frequency range (DCORSELx), the target DCO frequency, f_{DCO}, should be set to reside within the range of $f_{DCO(n,\ 0),MAX} \le f_{DCO} \le f_{DCO(n,\ 31),MIN}$, where $f_{DCO(n,\ 0),MAX}$ represents the maximum frequency specified for the DCO frequency, range n, tap 0 (DCOx = 0) and $f_{DCO(n,31),MIN}$ represents the minimum frequency specified for the DCO frequency, range n, tap 31 (DCOx = 31). This ensures that the target DCO frequency resides within the range selected. It should also be noted that if the actual f_{DCO} frequency for the selected range causes the FLL or the application to select tap 0 or 31, the DCO fault flag is set to report that the selected range is at its minimum or maximum tap setting.

(2) Calculated using the box method: (MAX(–40°C to 85°C) – MIN(–40°C to 85°C)) / MIN(–40°C to 85°C) / (85°C – (–40°C))

(3) Calculated using the box method: (MAX(1.8 V to 3.6 V) – MIN(1.8 V to 3.6 V)) / MIN(1.8 V to 3.6 V) / (3.6 V – 1.8 V)