CENG336

Int. to Embedded Systems Development

Recitation 2

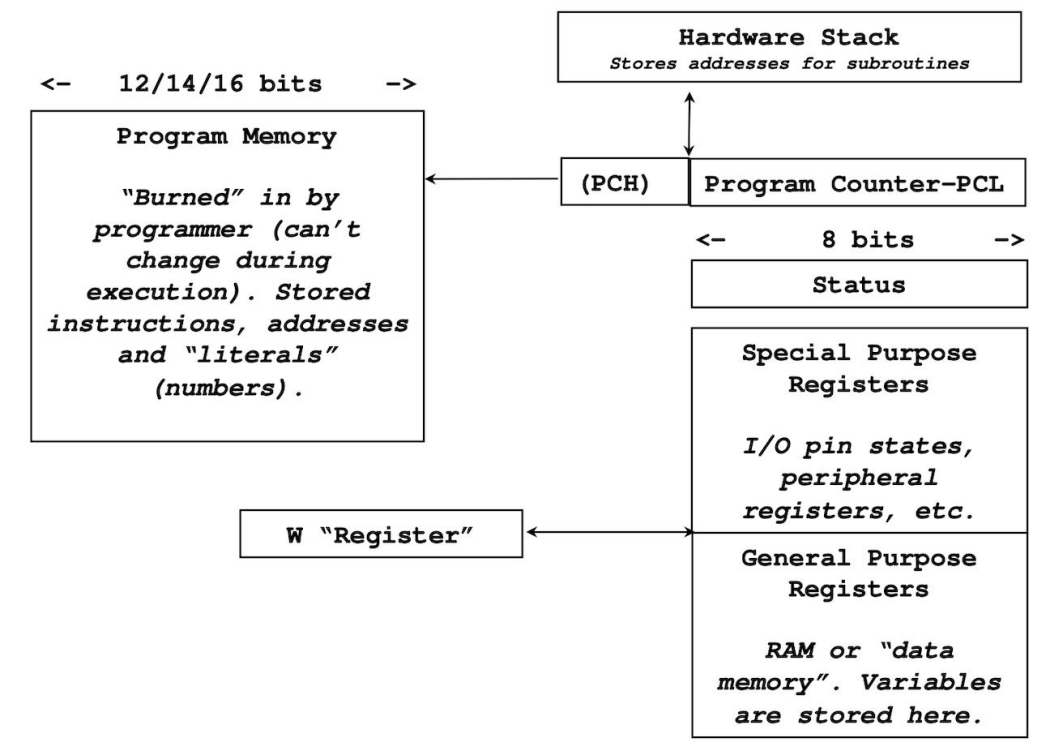
PIC Assembly Language &
I/O Ports of PIC18F8722

2021-2022 Spring

Outline

- PIC18F8722 overview
- PIC assembly language
  - code structure
  - instruction set
- I/O ports of PIC18F8722

PIC18F8722 overview: programmer's model



from Week 2 Architecture Lecture Notes in ODTUClass

PIC assembly language

PIC assembly language: code structure

```
PROCESSOR 18F8722

#include <xc.inc>

CONFIG OSC = HSPLL, FCMEN = OFF, IESO = OFF, ...

; global variable definitions
GLOBAL var1

; allocating space in access bank for variables
PSECT udata_acs
var1:
    DS 1     ; allocate 1 byte

PSECT resetVec,class=CODE,reloc=2
resetVec:
    goto    main

init:
    ...
task_1:
    ...

PSECT CODE
main:
    call init
    call task_1
    ...

end resetVec
```

specifying the processor

register names and other definitions

configurations

user-defined global variables

allocating space for user-defined variables in the access bank

setting the reset vector **`resetVec`** to mark the start of the program

user-defined code sections to be called in the main program (~ function definitions)

main code, where we set the **`resetVec`** to start from

end of the program

to avoid confusion...

currently: XC8 PIC Assembler
before: MPASM

- lecture notes and videos may include MPASM codes
- differences in:
  - setting the reset vector
  - variable definitions
  - some directives
  - some literal representations

PIC Assembler

```
PROCESSOR 18F8722

#include <xc.inc>

CONFIG OSC = HSPLL, FCMEN = OFF, IESO = OFF, ...

; global variable definitions
GLOBAL var1

; allocating space in access bank for variables
PSECT udata_acs
var1:
    DS 1     ; allocate 1 byte

PSECT resetVec,class=CODE,reloc=2
resetVec:
    goto    main

init:
    ...
task_1:
    ...

PSECT CODE
main:
    call init
    call task_1
    ...

end resetVec
```

MPASM (old)

```
LIST p=18F8722

#include <p18f8722.inc>

CONFIG OSC = HSPLL, FCMEN = OFF, IESO = OFF, ...

; variable definitions

...

ORG   0x00
goto  main

init:
    ...
task_1:
    ...

main:
    call init
    call task_1
    ...

end
```

instruction set

- ***byte-oriented*** operations
- ***bit-oriented*** operations
- ***literal*** operations
- ***control*** operations
- program memory ⇔ data memory operations

byte-oriented operations

TABLE 26-2: PIC18FXXXX INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb			LSb	Status Affected	Notes
BYTE-ORIENTED OPERATIONS									
ADDWF	f, d, a	Add WREG and f	1	0010	01da	ffff	ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and Carry bit to f	1	0010	00da	ffff	ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG with f	1	0001	01da	ffff	ffff	Z, N	1,2
CLRF	f, a	Clear f	1	0110	101a	ffff	ffff	Z	2
COMF	f, d, a	Complement f	1	0001	11da	ffff	ffff	Z, N	1, 2
CPFSEQ	f, a	Compare f with WREG, Skip =	1 (2 or 3)	0110	001a	ffff	ffff	None	4
CPFSGT	f, a	Compare f with WREG, Skip >	1 (2 or 3)	0110	010a	ffff	ffff	None	4
CPFSLT	f, a	Compare f with WREG, Skip <	1 (2 or 3)	0110	000a	ffff	ffff	None	1, 2
DECF	f, d, a	Decrement f	1	0000	01da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
DECFSZ	f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010	11da	ffff	ffff	None	1, 2, 3, 4
DCFSNZ	f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100	11da	ffff	ffff	None	1, 2
INCF	f, d, a	Increment f	1	0010	10da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
INCFSZ	f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011	11da	ffff	ffff	None	4
INFSNZ	f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100	10da	ffff	ffff	None	1, 2
IORWF	f, d, a	Inclusive OR WREG with f	1	0001	00da	ffff	ffff	Z, N	1, 2
MOVF	f, d, a	Move f	1	0101	00da	ffff	ffff	Z, N	1
MOVFF	f_s, f_d	Move f_s (source) to 1st word	2	1100	ffff	ffff	ffff	None	
		f_d (destination) 2nd word		1111	ffff	ffff	ffff		
MOVWF	f, a	Move WREG to f	1	0110	111a	ffff	ffff	None	
MULWF	f, a	Multiply WREG with f	1	0000	001a	ffff	ffff	None	1, 2
NEGF	f, a	Negate f	1	0110	110a	ffff	ffff	C, DC, Z, OV, N	
RLCF	f, d, a	Rotate Left f through Carry	1	0011	01da	ffff	ffff	C, Z, N	1, 2
RLNCF	f, d, a	Rotate Left f (No Carry)	1	0100	01da	ffff	ffff	Z, N	
RRCF	f, d, a	Rotate Right f through Carry	1	0011	00da	ffff	ffff	C, Z, N	
RRNCF	f, d, a	Rotate Right f (No Carry)	1	0100	00da	ffff	ffff	Z, N	
SETF	f, a	Set f	1	0110	100a	ffff	ffff	None	1, 2
SUBFWB	f, d, a	Subtract f from WREG with Borrow	1	0101	01da	ffff	ffff	C, DC, Z, OV, N	
SUBWF	f, d, a	Subtract WREG from f	1	0101	11da	ffff	ffff	C, DC, Z, OV, N	1, 2
SUBWFB	f, d, a	Subtract WREG from f with Borrow	1	0101	10da	ffff	ffff	C, DC, Z, OV, N	
SWAPF	f, d, a	Swap Nibbles in f	1	0011	10da	ffff	ffff	None	4
TSTFSZ	f, a	Test f, Skip if 0	1 (2 or 3)	0110	011a	ffff	ffff	None	1, 2
XORWF	f, d, a	Exclusive OR WREG with f	1	0001	10da	ffff	ffff	Z, N	

Reading and/or updating one byte of data, namely a register **f**

bit-oriented operations

TABLE 26-2: PIC18FXXXX INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb			LSb	Status Affected	Notes
BIT-ORIENTED OPERATIONS									
BCF	f, b, a	Bit Clear f	1	1001	bbba	ffff	ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000	bbba	ffff	ffff	None	1, 2
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	ffff	None	3, 4
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010	bbba	ffff	ffff	None	3, 4
BTG	f, b, a	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	
BN	n	Branch if Negative	1 (2)	1110	0110	nnnn	nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110	0011	nnnn	nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110	0111	nnnn	nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110	0101	nnnn	nnnn	None	
BNZ	n	Branch if Not Zero	1 (2)	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	2	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	n, s	Call Subroutine 1st word	2	1110	110s	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000	0000	0000	0100	$\overline{TO}$, $\overline{PD}$	
DAW	—	Decimal Adjust WREG	1	0000	0000	0000	0111	C	
GOTO	n	Go to Address 1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation	1	1111	xxxx	xxxx	xxxx	None	4
POP	—	Pop Top of Return Stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push Top of Return Stack (TOS)	1	0000	0000	0000	0101	None	
RCALL	n	Relative Call	2	1101	1nnn	nnnn	nnnn	None	
RESET		Software Device Reset	1	0000	0000	1111	1111	All	
RETFIE	s	Return from Interrupt Enable	2	0000	0000	0001	000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with Literal in WREG	2	0000	1100	kkkk	kkkk	None	
RETURN	s	Return from Subroutine	2	0000	0000	0001	001s	None	
SLEEP	—	Go into Standby mode	1	0000	0000	0000	0011	$\overline{TO}$, $\overline{PD}$	

Reading and/or updating one bit of data, or making a control decision using the read bit

control operations

TABLE 26-2: PIC18FXXXX INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb			LSb	Status Affected	Notes
BIT-ORIENTED OPERATIONS									
BCF	f, b, a	Bit Clear f	1	1001	bbba	ffff	ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000	bbba	ffff	ffff	None	1, 2
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	ffff	None	3, 4
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010	bbba	ffff	ffff	None	3, 4
BTG	f, b, a	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	
BN	n	Branch if Negative	1 (2)	1110	0110	nnnn	nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110	0011	nnnn	nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110	0111	nnnn	nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110	0101	nnnn	nnnn	None	
BNZ	n	Branch if Not Zero	1 (2)	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	2	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	n, s	Call Subroutine 1st word	2	1110	110s	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000	0000	0000	0100	$\overline{TO}$, $\overline{PD}$	
DAW	—	Decimal Adjust WREG	1	0000	0000	0000	0111	C	
GOTO	n	Go to Address 1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation	1	1111	xxxx	xxxx	xxxx	None	4
POP	—	Pop Top of Return Stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push Top of Return Stack (TOS)	1	0000	0000	0000	0101	None	
RCALL	n	Relative Call	2	1101	1nnn	nnnn	nnnn	None	
RESET		Software Device Reset	1	0000	0000	1111	1111	All	
RETFIE	s	Return from Interrupt Enable	2	0000	0000	0001	000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with Literal in WREG	2	0000	1100	kkkk	kkkk	None	
RETURN	s	Return from Subroutine	2	0000	0000	0001	001s	None	
SLEEP	—	Go into Standby mode	1	0000	0000	0000	0011	$\overline{TO}$, $\overline{PD}$	

Updating the control flow such as making calls, returns or branching

literal operations

TABLE 26-2: PIC18FXXXX INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb			LSb	Status Affected	Notes
LITERAL OPERATIONS									
ADDLW	k	Add Literal and WREG	1	0000	1111	kkkk	kkkk	C, DC, Z, OV, N	
ANDLW	k	AND Literal with WREG	1	0000	1011	kkkk	kkkk	Z, N	
IORLW	k	Inclusive OR Literal with WREG	1	0000	1001	kkkk	kkkk	Z, N	
LFSR	f, k	Move Literal (12-bit) 2nd word	2	1110	1110	00ff	kkkk	None	
		to FSR(f) 1st word		1111	0000	kkkk	kkkk		
MOVLB	k	Move Literal to BSR<3:0>	1	0000	0001	0000	kkkk	None	
MOVLW	k	Move Literal to WREG	1	0000	1110	kkkk	kkkk	None	
MULLW	k	Multiply Literal with WREG	1	0000	1101	kkkk	kkkk	None	
RETLW	k	Return with Literal in WREG	2	0000	1100	kkkk	kkkk	None	
SUBLW	k	Subtract WREG from Literal	1	0000	1000	kkkk	kkkk	C, DC, Z, OV, N	
XORLW	k	Exclusive OR Literal with WREG	1	0000	1010	kkkk	kkkk	Z, N	
DATA MEMORY ↔ PROGRAM MEMORY OPERATIONS									
TBLRD*		Table Read	2	0000	0000	0000	1000	None	
TBLRD*+		Table Read with Post-Increment		0000	0000	0000	1001	None	
TBLRD*-		Table Read with Post-Decrement		0000	0000	0000	1010	None	
TBLRD+*		Table Read with Pre-Increment		0000	0000	0000	1011	None	
TBLWT*		Table Write	2	0000	0000	0000	1100	None	5
TBLWT*+		Table Write with Post-Increment		0000	0000	0000	1101	None	5
TBLWT*-		Table Write with Post-Decrement		0000	0000	0000	1110	None	5
TBLWT+*		Table Write with Pre-Increment		0000	0000	0000	1111	None	5

Using a literal **k** instead of a value stored in a register

memory operations

TABLE 26-2: PIC18FXXXX INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb			LSb	Status Affected	Notes
LITERAL OPERATIONS									
ADDLW	k	Add Literal and WREG	1	0000	1111	kkkk	kkkk	C, DC, Z, OV, N	
ANDLW	k	AND Literal with WREG	1	0000	1011	kkkk	kkkk	Z, N	
IORLW	k	Inclusive OR Literal with WREG	1	0000	1001	kkkk	kkkk	Z, N	
LFSR	f, k	Move Literal (12-bit) 2nd word	2	1110	1110	00ff	kkkk	None	
		to FSR(f) 1st word		1111	0000	kkkk	kkkk		
MOVLB	k	Move Literal to BSR<3:0>	1	0000	0001	0000	kkkk	None	
MOVLW	k	Move Literal to WREG	1	0000	1110	kkkk	kkkk	None	
MULLW	k	Multiply Literal with WREG	1	0000	1101	kkkk	kkkk	None	
RETLW	k	Return with Literal in WREG	2	0000	1100	kkkk	kkkk	None	
SUBLW	k	Subtract WREG from Literal	1	0000	1000	kkkk	kkkk	C, DC, Z, OV, N	
XORLW	k	Exclusive OR Literal with WREG	1	0000	1010	kkkk	kkkk	Z, N	
DATA MEMORY ↔ PROGRAM MEMORY OPERATIONS									
TBLRD*		Table Read	2	0000	0000	0000	1000	None	
TBLRD*+		Table Read with Post-Increment		0000	0000	0000	1001	None	
TBLRD*-		Table Read with Post-Decrement		0000	0000	0000	1010	None	
TBLRD+*		Table Read with Pre-Increment		0000	0000	0000	1011	None	
TBLWT*		Table Write	2	0000	0000	0000	1100	None	5
TBLWT*+		Table Write with Post-Increment		0000	0000	0000	1101	None	5
TBLWT*-		Table Write with Post-Decrement		0000	0000	0000	1110	None	5
TBLWT+*		Table Write with Pre-Increment		0000	0000	0000	1111	None	5

read and write data between data memory and program memory

structure of an instruction in datasheet

ADDWF	ADD W to f
Syntax:	ADDWF f {,d {,a}}
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(W) + (f) → dest
Status Affected:	N, OV, C, DC, Z
Encoding:	0010 | 01da | ffff | ffff
Description:	Add W to register 'f'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f' (default). If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).

since f is an address, you can directly write a variable with and allocated space as an operand

bits in the STATUS register that are affected by the execution of this instruction

ADDWF: add W and F

ADDWF	ADD W to f
Syntax:	ADDWF f {,d {,a}}
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(W) + (f) → dest
Status Affected:	N, OV, C, DC, Z
Encoding:	0010 | 01da | ffff | ffff
Description:	Add W to register 'f'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f' (default). If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).

add the values of working register and a register **f**

- `addwf my_var, 0`
  store result back in `W`
- `addwf my_var, 1`
  store result back in `my_var`

CLRF: clear F

CLRF	Clear f
Syntax:	CLRF f {,a}
Operands:	$0 \le f \le 255$ $a \in [0,1]$
Operation:	$000h \rightarrow f$ $1 \rightarrow Z$
Status Affected:	Z
Encoding:	0110 101a ffff ffff
Description:	Clears the contents of the specified register.

clear the contents of register **f**

- before:
  `my_var => 00010111`
- after `clrf my_var`:
  `my_var => 00000000`

move operations

- `movf f, 0` : moves value of f to W
- `movwf f` : moves value of W to f
- `movlw k` : moves literal k to W
- `movff fs, fd` : moves value of fs to fd

DECF and INCF: decrement F and increment F

DECF	Decrement f
Syntax:	DECF f {,d {,a}}
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(f) – 1 → dest
Status Affected:	C, DC, N, OV, Z
Encoding:	0000 | 01da | ffff | ffff
Description:	Decrement register 'f'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f' (default).

INCF	Increment f
Syntax:	INCF f {,d {,a}}
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(f) + 1 → dest
Status Affected:	C, DC, N, OV, Z
Encoding:	0010 | 10da | ffff | ffff
Description:	The contents of register 'f' are incremented. If 'd' is '0', the result is placed in W. If 'd' is '1', the result is placed back in register 'f' (default).

- **`decf f, 0:`**
  subtract 1 from the value of f,
  write the result to W
- **`decf f, 1:`**
  write the result to f instead

- **`incf f, 0:`**
  add 1 to the value of f,
  write the result to W
- **`incf f, 1:`**
  write the result to f instead

DECFSZ and INCFSZ: dec./inc., skip if zero

- **`decfsz f, 1`**
  decrement f
  if result is 0, skip the next instruction

- **`incfsz f, 1`**
  increment f
  if result is 0, skip the next instruction

d set to 1:
write back to f

```
nonzero_case:
        ...

zero_case:
        ...

main:
        ...
        decfsz my_var, 1
        goto nonzero_case
        goto zero_case
```

NOP: no operation

NOP	No Operation
Syntax:	NOP
Operands:	None
Operation:	No operation
Status Affected:	None
Encoding:	0000 1111 | 0000 xxxx | 0000 xxxx | 0000 xxxx
Description:	No operation.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation

Example:

None.

stands for **no** o**p**eration, does nothing

BCF and BSF: bit clear F and bit set F

- **`bcf f, b`**
  set bit **b** of register **f** to 0
  (clear bit **b** of register **f**)

- **`bsf f, b`**
  set bit **b** of register **f** to 1

before:

```
my_var => 00010111
```

after `bcf my_var, 2`:

```
my_var => 00010011
```

BTFSC and BTFSS: bit test f, skip if clear/set

- **`btfsc f, b`**
  test bit **b** of register **f**
  if bit is 0, skip the next instruction

- **`btfss f, b`**
  test bit **b** of register **f**
  if bit is 1, skip the next instruction

```
nonzero_case:
      ...

zero_case:
      ...

main:
      ...
      btfsc my_var, 1
      goto nonzero_case
      goto zero_case
```

CALL and RETURN

`call my_label`

- return address PC+4 is pushed onto the stack
- the PC is loaded with the address of the label

`return`

- stack is popped into the PC, transferring control to the instruction after the original call.

CALL and RETURN

```
init:
    ...
    return
task_1:
    ...
    return

main:
    call init
    call task_1
    ...
```

- `init` is called
- address of `call task_1` is pushed to stack
- address of `init` is loaded to PC
- `init` executes
- `init` returns
- address of `call task_1` is popped from the stack to PC

I/O ports of PIC18F8722

I/O ports

For ports A-J on the device:

PORTX: register that reads the voltage levels on the pins of the port

TRISX: register for setting the *data direction* of the port

a bit is **set**:	**1**NPUT mode
a bit is cleared:	**0**UTPUT mode

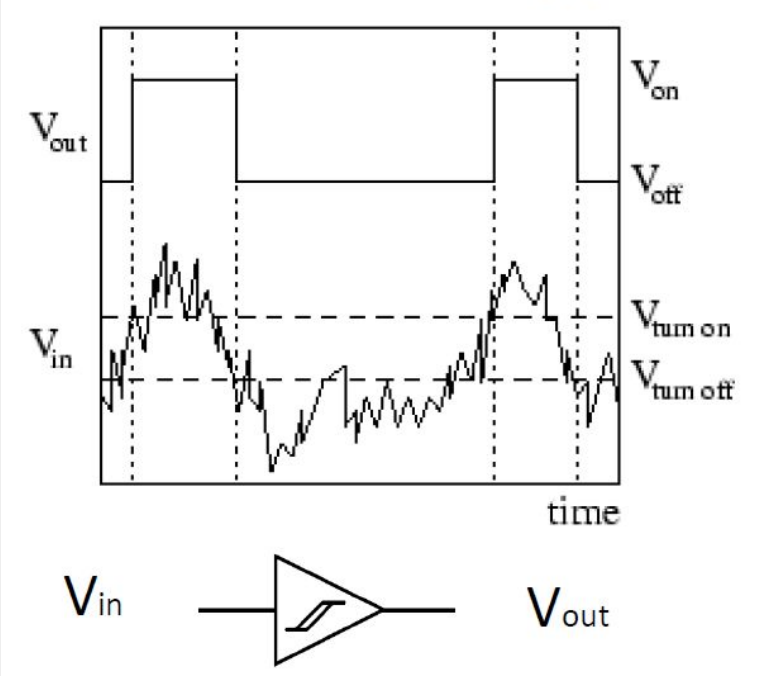
LATX: *data latch* register of the port, also called output latch

special features of I/O ports

- most basic features of ports are:
  - turning on/off the leds
  - reflecting the state changes happening when a button is pressed/released
- most ports have additional features:
  - oscillator connection
  - interrupts
  - analog input

    ...
- no need to memorize, just check the datasheet!

special features: Schmitt Trigger buffer



prevents the vibrations that occur when state change happens in a port (like pressing or releasing button).

PORTA

8-bit wide, bidirectional port

data direction register: **TRISA**

data latch register: **LATA**

some special features:

RA4 pin: read as Timer0 clock input in counter mode (RA4/T0CKI)
has a Schmitt trigger input buffer

RA6:RA7 pins: can be enabled as oscillator

RA5:RA0 pins: can operate as A/D converter inputs

note: on a power-on reset, all pins in PORTA are configured as analog inputs and read as ‘0’, except RA4, which is configured as a digital input.

PORTB

8-bit wide, bidirectional port

data direction register: **TRISB**

data latch register: **LATB**

some special features:

RB4:RB7 pins: can be configured as interrupt pins

- a change in any of these pins will trigger an interrupt and set the RBIF bit in INTCON register
- clearing RBIF, or reading/writing***** to PORTB will clear the interrupt

***** check datasheet for exceptions

PORTC

8-bit wide, bidirectional port

data direction register: **TRISC**

data latch register: **LATC**

some special features:

- Schmitt trigger input buffers on all pins

note: on a power-on reset, all pins are configured as digital inputs.

PORTD

8-bit wide, bidirectional port

data direction register: **TRISD**

data latch register: **LATD**

some special features:

- Schmitt trigger input buffers on all pins

note: on a power-on reset, all pins are configured as digital inputs.

PORTE

8-bit wide, bidirectional port

data direction register: **TRISE**

data latch register: **LATE**

some special features:

- Schmitt trigger input buffers on all pins

note: on a power-on reset, all pins are configured as digital inputs.

PORTF

8-bit wide, bidirectional port

data direction register: **TRISF**

data latch register: **LATF**

some special functions:

- Schmitt trigger input buffers on all pins
- **RF6:RF0 pins**: can operate as A/D converter inputs

notes: on a power-on reset, the RF<6:0> pins are configured as analog inputs and read as ‘0’. ADCON1 register can be set to to configure PORTF as digital I/O.

PORTG

6-bit wide, bidirectional port

data direction register: **TRISG**

data latch register: **LATG**

some special functions:

- Schmitt trigger input buffers on all pins
- **RG5 pin:** an input-only pin

note: on a power-on reset, all pins are configured as digital inputs, but RG5 may be affected by some configurations.*****

***** check datasheet for details

PORTH

8-bit wide, bidirectional port

data direction register: **TRISH**

data latch register: **LATH**

some special functions:

- Schmitt trigger input buffers on all pins

note: on a power-on reset, all pins are configured as digital inputs.

PORTJ

8-bit wide, bidirectional port

data direction register: **TRISJ**

data latch register: **LATJ**

some special functions:

Schmitt trigger input buffers on all pins

note: on a power-on reset, all pins are configured as digital inputs.

!!! PORTJ is used as a data register for 7-segment display in MCDEV.

takeaways

takeaways

- datasheets and guides shared in ODTUClass are your friends!
  - architecture, instruction set and i/o ports: **PIC18F8722 DataSheet**
  - program structure: **MPLAB XC8 PIC Assembler User Guide**
- lecture notes and videos have detailed explanations that worth checking again
- you can <u>always</u> ask your questions via email or in office hours