week 2 - PIC 18F architecture

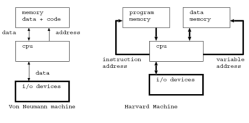
PIC microcontrollers = programmable interface controllers are electronic circuits that can be programmed to carry out vast range of tasks

↳ 8-bit: PIC10, PIC12, PIC16, PIC18 (some 16bit) 16-bit: PIC24F, PIC24H, dsPIC30, dsPIC33 32-bit: PIC32

↳ harvard architecture, RISC, small instruction set, cheap, high clock speed

harvard architecture = simpler, developed to overcome the bottleneck of Von Neumann, more costly

- faster, because separate program bus and data bus → greater flow of data is possible
- CPU can access instructions and read/write at the same time (von neumann can't → bottleneck)
- executes any instruction using only one clock cycle (two in von neumann)



The Microchip PIC Family

Architecture	Family		Data Width	Instruction Width
8-bit MCU	PIC10 / PIC12 / PIC16	Baseline	8 bits	12-bits
		Mid-Range		14-bits
	PIC18			16-bits
16-bit MCU	PIC24		16 bits	16 bits
	dsPIC30	Integrated DSP		
32-bit MCU			32 bits	32 bits

→ all instructors are one word

data width = wider integer → higher precision arithmetic

instruction width = wider → more complex instructions + higher precision arithmetic

the PIC family

↳ program memory

↳ flash = re-writeable, much faster to develop on

↳ eprom = erasable programmable read only memory

↳ hard to write-erase

↳ control registers uses special func reg to control

↳ peripherals = a device connected to a comp to provide communication (i/o) or func., PIC's have them too

Sample PIC Families

	PIC10Fxx	PIC12F5xx	PIC16F5xx	PIC10F3xx PIC12F6xx PIC16F6xx	PIC18F5xx
Instruction word	12 bits	12 bits	12 bits	14 bits	16 bits
Instructions	33	33	33	35	83
Program memory	256 – 512 words	512 – 1024 words	1024 – 2048 words	256 – 8192 words	2 Kwords – 64 Kwords
ROM	Flash	Flash	Flash	Flash	Flash
Data memory (bytes)	16 – 24	25 – 41	25 – 134	56 – 368	256 – 4K
Interrupts	0	0	0	int / ext	int / ext
Pins	6	8	14 – 40	6 – 64	18 – 100
I/O pins	4	6	12 – 32	4 – 54	16 – 70
Stack	2 levels	2 levels	2 levels	8 levels	31 levels
Timers	1	1	1	2 – 3	2 – 5
Bulk price	$0.35	$0.50	$0.50 – $0.85	$0.35 – $2.50	$1.20 - $8.50

PIC18 features

↳ accumulator based: stores the intermediate results of calculations in a special register, faster

↳ harvard arch, 75+8 inst with length 16-bit, 31×21 bit hardware stack

↳ speed: non-branching: 1 (1×4), branching: 2 (2×4) (cyc/inst) — an instruction cycle is four clock cycles

PIC18 programmer's model

↳ ALU and the core + special func registers (SFRs) from data memory

↳ registers: WREG: 8-bit working register (accumulator, stores intermediate result)

↳ BSR: bank select register

↳ STATUS: flag register (stores result) — flags: C, DC, Z, OV, N

↳ PC, Table Pointer, SP (stack pointer), Stack, SF

↳ memory: program mem (flash)

↳ data mem (general purpose register (GPR) files (RAM) + special purpose register (SPR) files) ↗ used at runtime

↳ 4 addressing modes: inherent = no argument, effecting device globally or operate one register implicitly. SLEEP, RESET, DAW

↳ literal/immediate = with additional explicit argument (literal)

CALL, GOTO, ADDLW, MOVLW → add/move a literal val to W register

↳ direct = specifices source or destination address, BSR

↳ indirect: 12 bit address is written one of the three special func reg FSRs (file select registers)

- one of INDFs, POSTINCx (increment), POSTDECx (decrement), PREINCx, PLUSx is used to get content of the address pointed by FSR (work as a pointer)
- usefull for manipulating data arrays located in GPR registers

PIC18 instruction set

instruction format: opcode | d | a | address

- ↳ access bit → determines what address means (use bank select reg or not)
- ↳ result destination → determines whether output from ALU goes to WREG or memory)
- ↳ arithmetic/logic function to be performed

↳ 77 instructions = 73 one word long (16 bit) + 4 two word long (32 bit) → divided into seven groups

1. move (data copy) and load
2. arithmetic
3. logic
4. program redirection (branch/jumps)
5. bit manipulation → movff, goto, call, lfsr → move literal to file select register
6. table read/write
7. machine control

banks = PIC18 4096 registers divided into 16 banks, only one bank is active at a time

↳ bank switching can cause errors → access bank to min errors

access bank = lower 96, upper/last 160 locations → a=0 no bank switching

- if no "a" in inst, to access special func reg set BSR to 0xF (bank 15)

movwf floc [,a] = write content of WREG (working reg) to floc (file registers / data mem) 8-bit

a=0 → use access bank (default) - ACCESS a=1 → use BSR - BANKED

- movwf 0x070,0 : (0x70) ← w
  movwf 0x070, ACCESS
  0110 1110 1111 0000 (opcode, 0, 0x070)
- movwf 0x070, 1 : BSR=0 → 0x070
  movwf 0x070, BANKED BSR=1 → 0x170
  BSR=2 → 0x270
  ⋮

floc = [0x000, 0x05F] ∪ [0xF60, 0xFFF] → a=0 ignore BSR

else → a=1 (movwf 0x070,1 } some BSR=0; movwf 0x070,0 } but never written)

movlb k = move 4-bit literal into BSR (16 bank → 4-bits) 0000 0001 0000 kkkk

movlb 2 : 0000 0001 0000 0010 (opcode, no need, select bank 2)

Bank select register is set to 2, to be used later on

movwf floc [,d[,a]] = d ← floc copies a val from data mem to w, or back to data mem

d=0 → w reg d=1 → f reg 0101 00da ffff ffff

- movf 0x01D, w w ← [0x01D]
- movf 0x01D, f [0x01D] ← [0x01D] → it is not useless, because it changes status bits

movff fs, fd = move contents of location fs to location fd 1100 ffff ffff ffff (src) 1100 ffff ffff ffff (dst)

- movff 0x1A0, 0x23F : [0x23F] ← [0x1A0]
- two words long

```
movlb  0x1    select bank 1
movf   0x1A0,w  w ← [0x1A0]      } long way
movlb  0x2    select bank 2
movwf  0x23F  [0x23F] ← w
```

2 cycles

addwf floc [,d[,a]] = d ← (floc) + (w)

addwf 0x070, w w ← [0x070] + (w) — reg changes

addwf 0x070, f [0x070] ← [0x070] + (w) — mem changes

subwf floc [,d[,a]] = d ← (floc) - (w)

subwf 0x070, w w ← [0x070] - (w)

subwf 0x070, f [0x070] ← [0x070] - (w)

op-code = movwf, addwf ...

source = W register, memory location (floc), literal

destination = w register, memory location (floc)

* floc/mem location in instructions are 8-bits ↳ need extra 4 bits from bank select register

0x01234567

- big endian = 01 23 45 67 (0x100 0x101 0x102 0x103) (standard)
- little endian = 67 45 23 01