CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP 122

DR JEFF SOFTWARE INDIE APP DEVELOPER

COMP122

Rev 8-11-21

ASSEMBLY Programming/ISA

MicroControllers

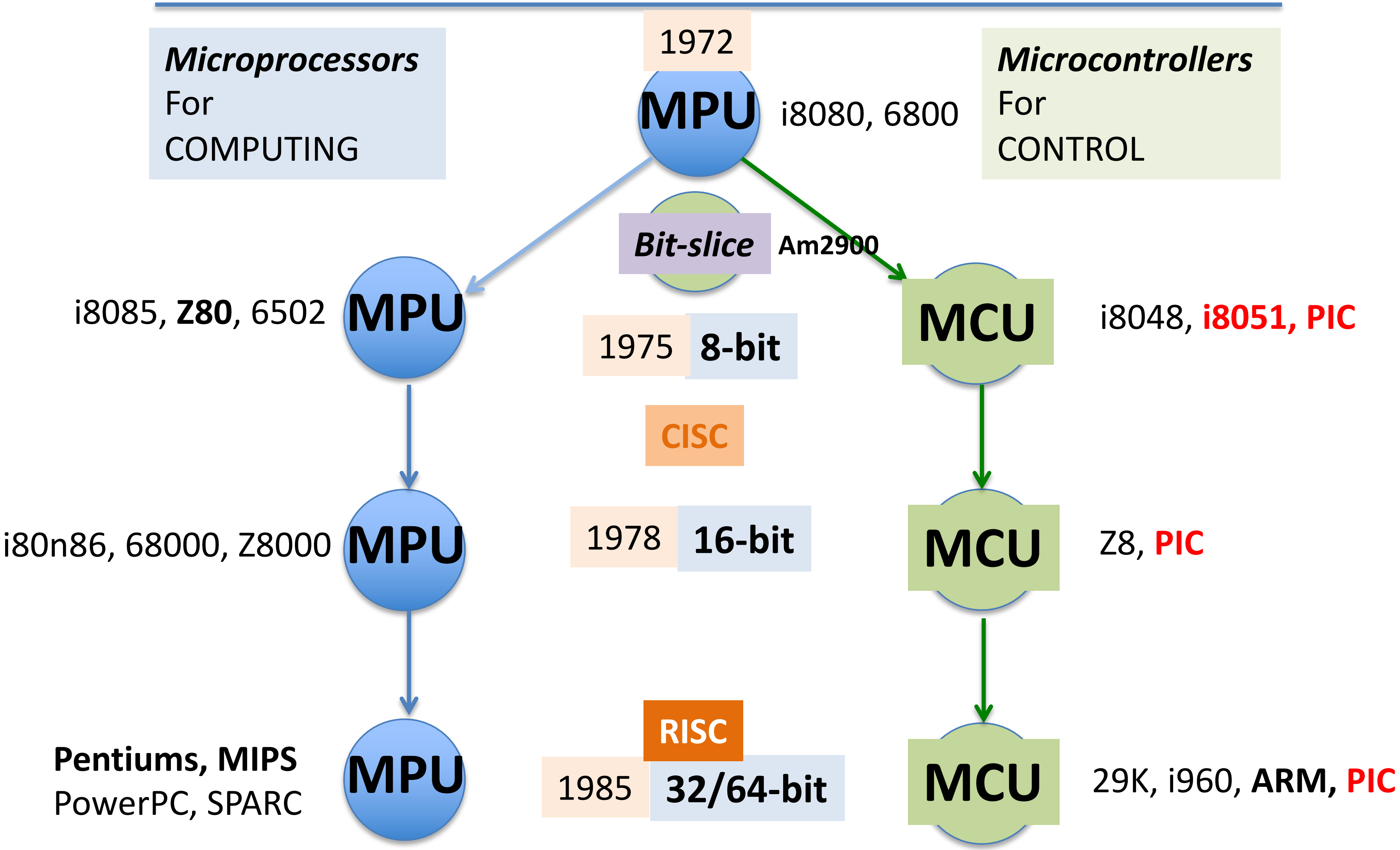
Dr Jeff Drobman

website → drjeffsoftware.com/classroom.html

email → jeffrey.drobman@csun.edu

Index

MPU/MCU Generations



Microprocessors
For
COMPUTING

Microcontrollers
For
CONTROL

1972 MPU i8080, 6800

Bit-slice **Am2900**

i8085, **Z80**, 6502 MPU

1975 **8-bit**

MCU i8048, **i8051, PIC**

CISC

i80n86, 68000, Z8000 MPU

1978 **16-bit**

MCU Z8, **PIC**

Pentiums, MIPS
PowerPC, SPARC
MPU

RISC
1985 **32/64-bit**

MCU 29K, i960, **ARM**, **PIC**

Embedded Control

Microprocessors
For
COMPUTING

- ❖ All 32/64-bit CPUs
- ❖ Large *data processing* applications
  - ◆ Employee records
  - ◆ Accounting
  - ◆ Payroll
- ❖ Operating systems (OS)
- ❖ "Apps" (applications)
  - ◆ PC/Mac
  - ◆ Mobile (phones, tablets)
  - ◆ Web apps
  - ◆ Cloud apps (SaaS)

Focus is ***Memory*** for large Data Files

Large DRAM, Disk, Flash

Microcontrollers
For
CONTROL

- ✧ *Real-time*
- ✧ *All-in-one*

- ❖ Small *embedded control* applications (8-bit MCU)
  - ◆ Appliances
  - ◆ Disk controllers
  - ◆ Remote controllers
  - ◆ Garage/gate openers

- ✧ Tiny
- ✧ Low power
- ✧ Low cost

- ❖ Medium *embedded control* (16-bit MCU)
  - ◆ User devices (iPods, phones, etc.)
  - ◆ Car/Airplane engine control
  - ◆ Car/Airplane braking & safety
  - ◆ Car transmission control
  - ◆ Home Automation (HAN)
- ❖ Large *embedded control* (32/64-bit MCU)
  - ◆ Car/Airplane entertainment
  - ◆ Car/Airplane navigation, systems management
  - ◆ Printers (MF)
  - ◆ Communications gear (WiFi, cable TV boxes)

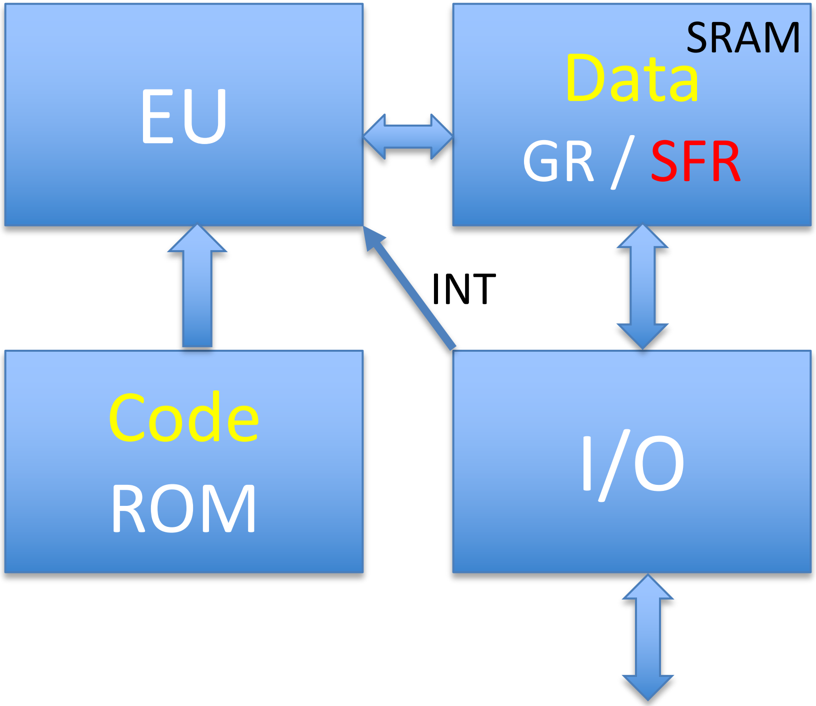
Focus is ***I/O*** – *Interrupts*

MCU Block Diagram

8/16/32-bit

DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2022

BASIC MODEL

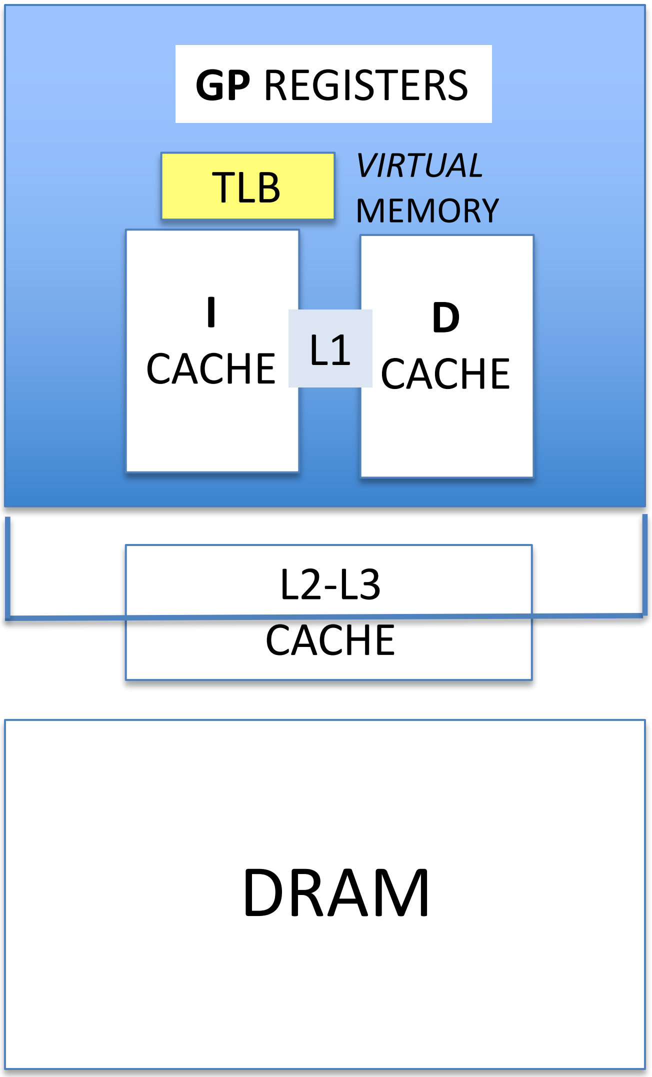


All on one cheap chip

- No Cache
- No External RAM

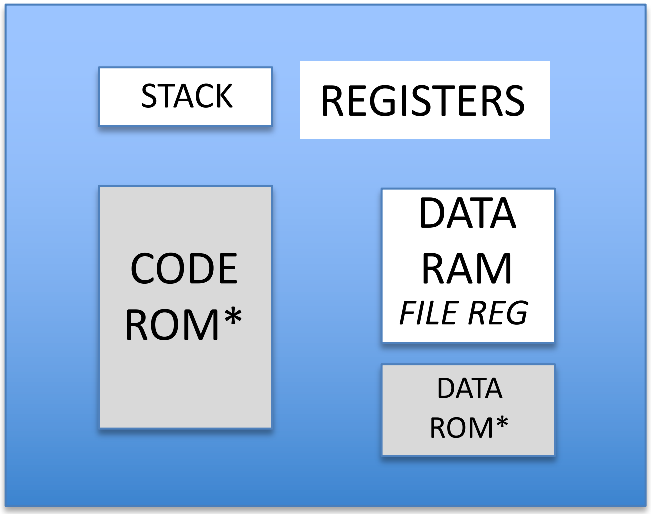
Memory: CPU vs MCU

MICRO**PROCESSOR**



LARGE
EXTERNAL
MEMORY

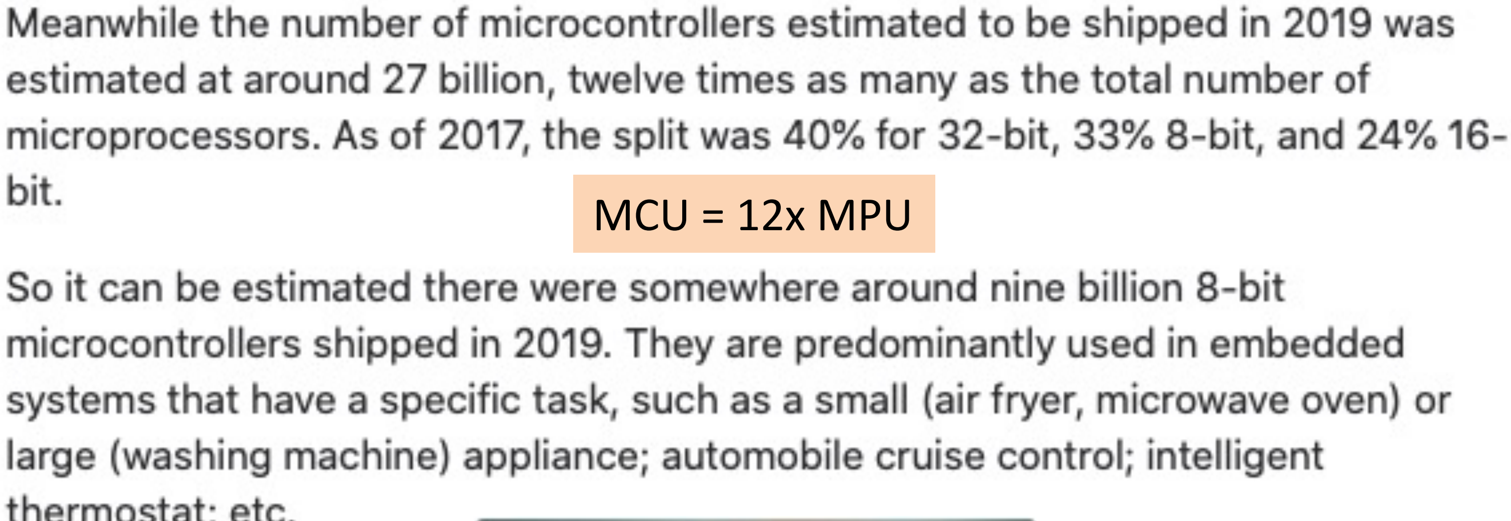
MICRO**CONTROLLER**



SMALL
INTERNAL
MEMORY

**ROM contents must be “programmed” “burned”, or masked*

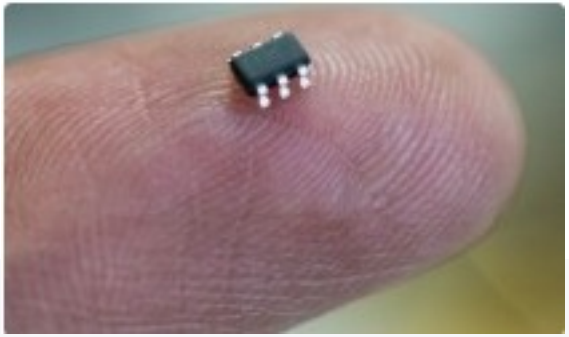
MCU

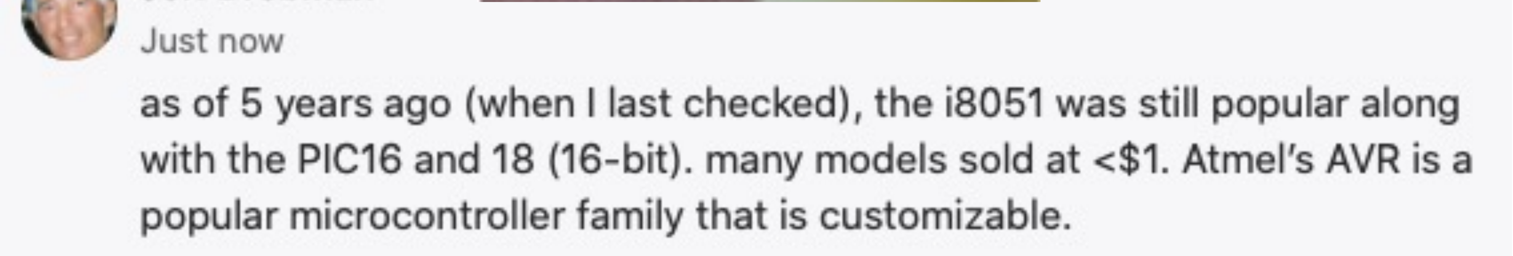


Meanwhile the number of microcontrollers estimated to be shipped in 2019 was estimated at around 27 billion, twelve times as many as the total number of microprocessors. As of 2017, the split was 40% for 32-bit, 33% 8-bit, and 24% 16-bit.

MCU = 12x MPU

So it can be estimated there were somewhere around nine billion 8-bit microcontrollers shipped in 2019. They are predominantly used in embedded systems that have a specific task, such as a small (air fryer, microwave oven) or large (washing machine) appliance; automobile cruise control; intelligent thermostat; etc.





Jeff Drobman

Just now

as of 5 years ago (when I last checked), the i8051 was still popular along with the PIC16 and 18 (16-bit). many models sold at <$1. Atmel's AVR is a popular microcontroller family that is customizable.

Multiply & Divide

MULTIPLY

- *Unsigned* only
- First convert negative numbers (2sC) – NEG op
- Compute result sign: 0 if both signs same, 1 else (not=)
- Complement result if sign is negative – NEG op
- Other MPUs use *signed* multiply (2sC) via “Booth’s Algorithm”

DIVIDE

- No hardware, no instruction
- Create subroutine (may find ones in asm library)
- Compute
  - Long division
  - Non-restoring division
  - Iterative subtraction (very slow)
- Use tricks
  - Divide by **2** or any **2^n**: right SHIFT by n
  - Divide by **10**: convert to BCD, then right SHIFT by 4 (reconvert to binary)
  - Divide by **5**: divide by 10, then multiply by 2 (by shifting after conv. Bin)

ISA

i8051

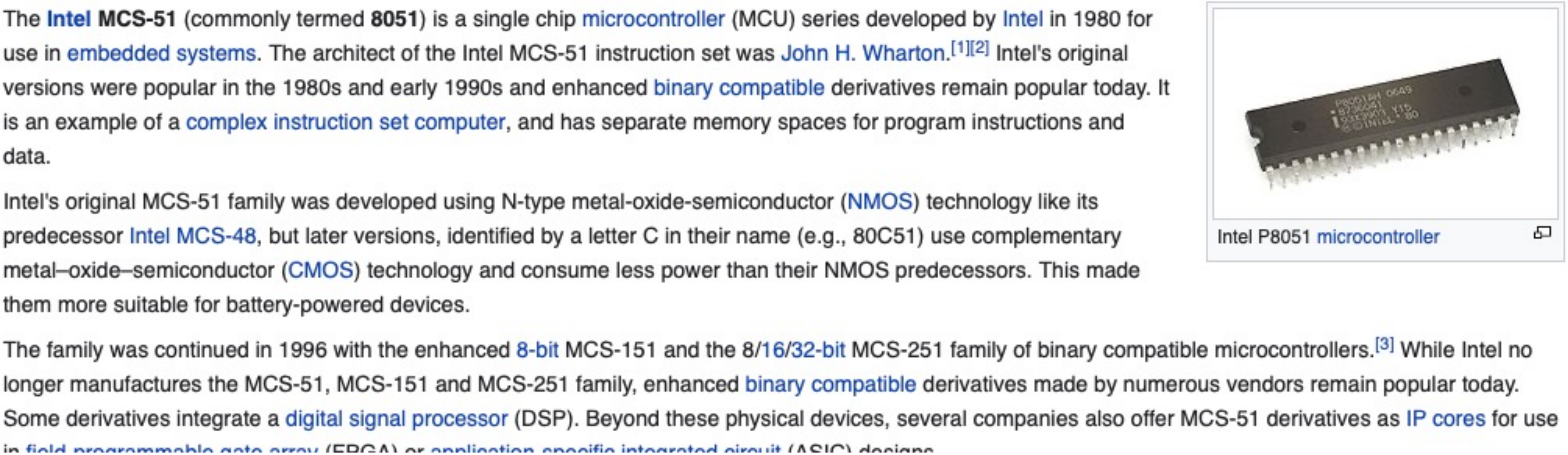
COMP122

i8051 MCU

MCS-51 1980

Intel 8051

From Wikipedia, the free encyclopedia



The **Intel MCS-51** (commonly termed **8051**) is a single chip microcontroller (MCU) series developed by Intel in 1980 for use in embedded systems. The architect of the Intel MCS-51 instruction set was John H. Wharton.[1][2] Intel's original versions were popular in the 1980s and early 1990s and enhanced binary compatible derivatives remain popular today. It is an example of a complex instruction set computer, and has separate memory spaces for program instructions and data.

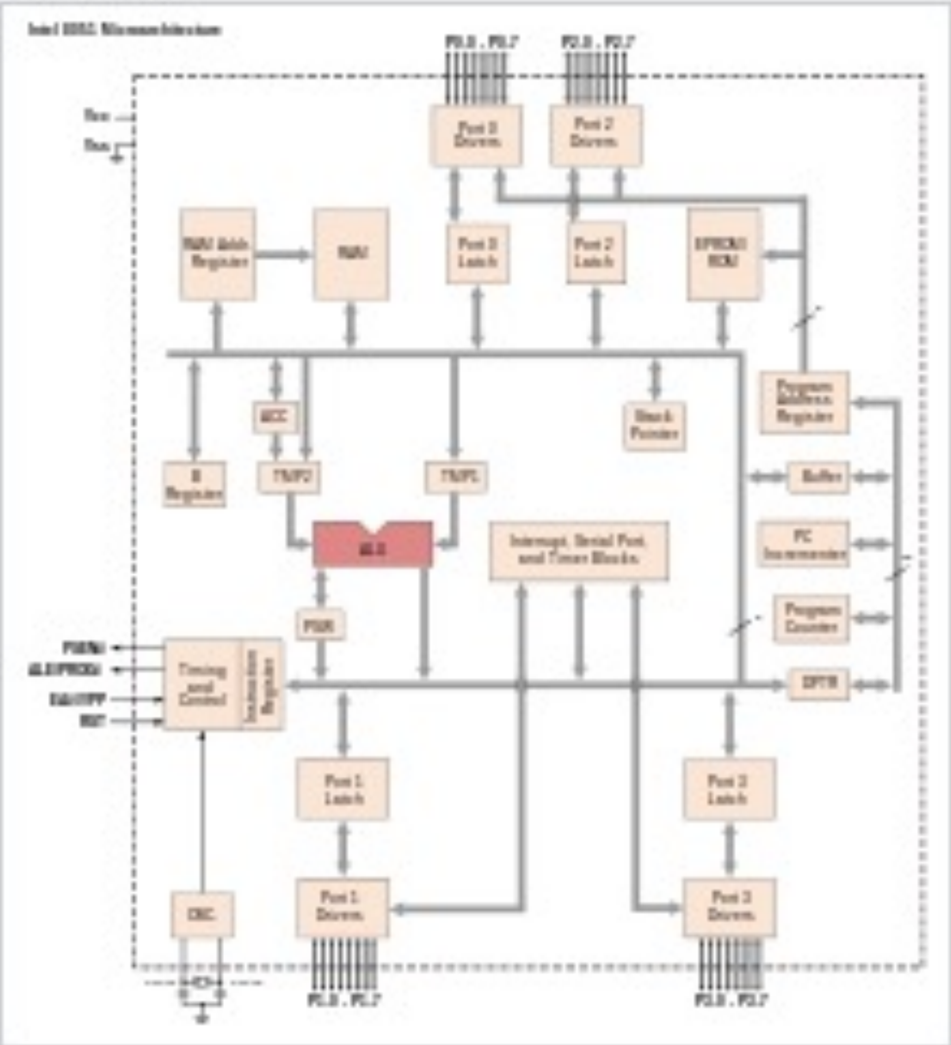
Intel P8051 microcontroller

Intel's original MCS-51 family was developed using N-type metal-oxide-semiconductor (NMOS) technology like its predecessor Intel MCS-48, but later versions, identified by a letter C in their name (e.g., 80C51) use complementary metal–oxide–semiconductor (CMOS) technology and consume less power than their NMOS predecessors. This made them more suitable for battery-powered devices.

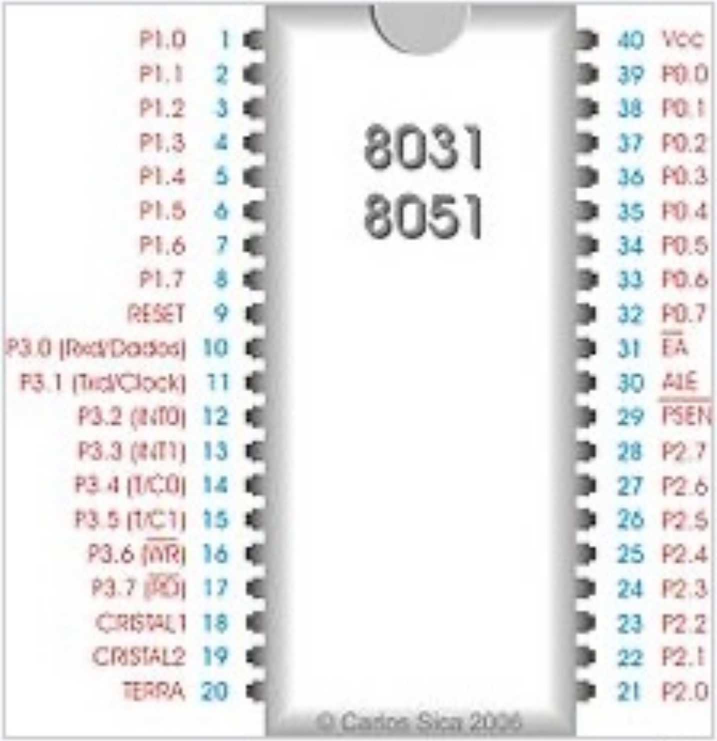
The family was continued in 1996 with the enhanced 8-bit MCS-151 and the 8/16/32-bit MCS-251 family of binary compatible microcontrollers.[3] While Intel no longer manufactures the MCS-51, MCS-151 and MCS-251 family, enhanced binary compatible derivatives made by numerous vendors remain popular today. Some derivatives integrate a digital signal processor (DSP). Beyond these physical devices, several companies also offer MCS-51 derivatives as IP cores for use in field programmable gate array (FPGA) or application-specific integrated circuit (ASIC) designs.

Intel P8051 microcontroller

i8051 MCU

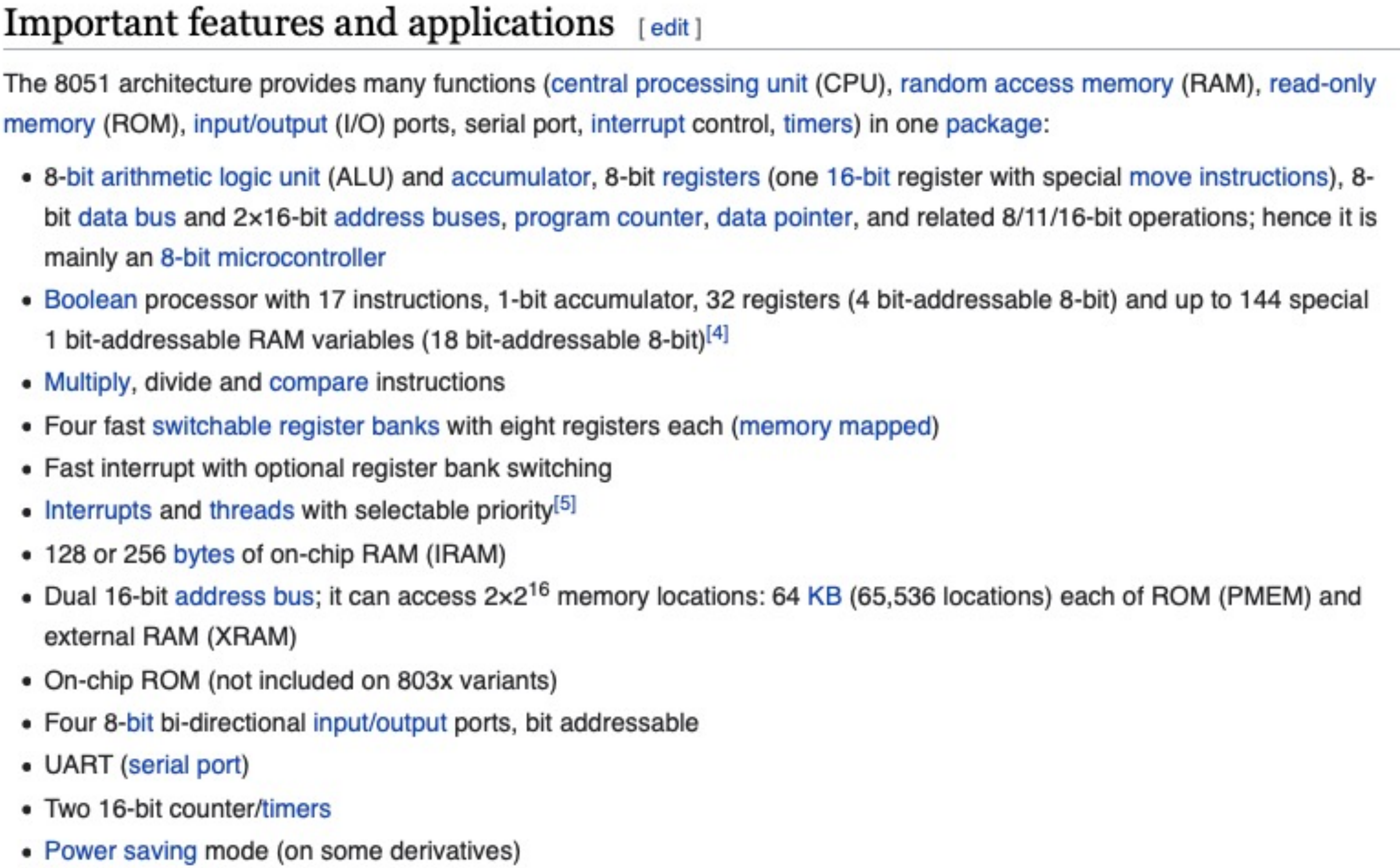


i8051 microarchitecture



i8051 pinout

i8051 MCU

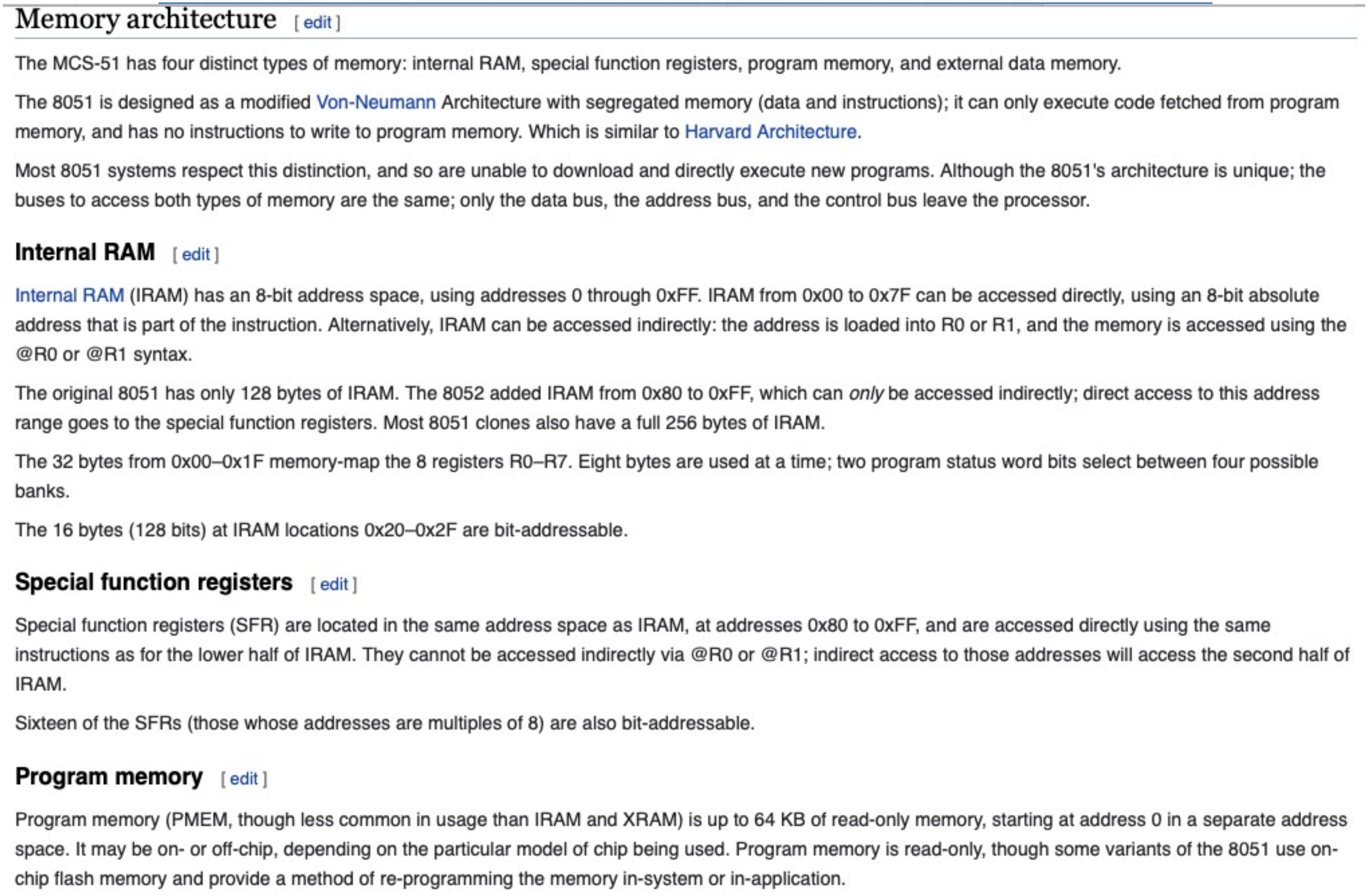


Important features and applications

The 8051 architecture provides many functions (central processing unit (CPU), random access memory (RAM), read-only memory (ROM), input/output (I/O) ports, serial port, interrupt control, timers) in one package:

- 8-bit arithmetic logic unit (ALU) and accumulator, 8-bit registers (one 16-bit register with special move instructions), 8-bit data bus and 2×16-bit address buses, program counter, data pointer, and related 8/11/16-bit operations; hence it is mainly an 8-bit microcontroller
- Boolean processor with 17 instructions, 1-bit accumulator, 32 registers (4 bit-addressable 8-bit) and up to 144 special 1 bit-addressable RAM variables (18 bit-addressable 8-bit)[4]
- Multiply, divide and compare instructions
- Four fast switchable register banks with eight registers each (memory mapped)
- Fast interrupt with optional register bank switching
- Interrupts and threads with selectable priority[5]
- 128 or 256 bytes of on-chip RAM (IRAM)
- Dual 16-bit address bus; it can access 2×2^{16} memory locations: 64 KB (65,536 locations) each of ROM (PMEM) and external RAM (XRAM)
- On-chip ROM (not included on 803x variants)
- Four 8-bit bi-directional input/output ports, bit addressable
- UART (serial port)
- Two 16-bit counter/timers
- Power saving mode (on some derivatives)

i8051 MCU



Memory architecture [edit]

The MCS-51 has four distinct types of memory: internal RAM, special function registers, program memory, and external data memory.

The 8051 is designed as a modified Von-Neumann Architecture with segregated memory (data and instructions); it can only execute code fetched from program memory, and has no instructions to write to program memory. Which is similar to Harvard Architecture.

Most 8051 systems respect this distinction, and so are unable to download and directly execute new programs. Although the 8051's architecture is unique; the buses to access both types of memory are the same; only the data bus, the address bus, and the control bus leave the processor.

Internal RAM [edit]

Internal RAM (IRAM) has an 8-bit address space, using addresses 0 through 0xFF. IRAM from 0x00 to 0x7F can be accessed directly, using an 8-bit absolute address that is part of the instruction. Alternatively, IRAM can be accessed indirectly: the address is loaded into R0 or R1, and the memory is accessed using the @R0 or @R1 syntax.

The original 8051 has only 128 bytes of IRAM. The 8052 added IRAM from 0x80 to 0xFF, which can *only* be accessed indirectly; direct access to this address range goes to the special function registers. Most 8051 clones also have a full 256 bytes of IRAM.

The 32 bytes from 0x00–0x1F memory-map the 8 registers R0–R7. Eight bytes are used at a time; two program status word bits select between four possible banks.

The 16 bytes (128 bits) at IRAM locations 0x20–0x2F are bit-addressable.

Special function registers [edit]

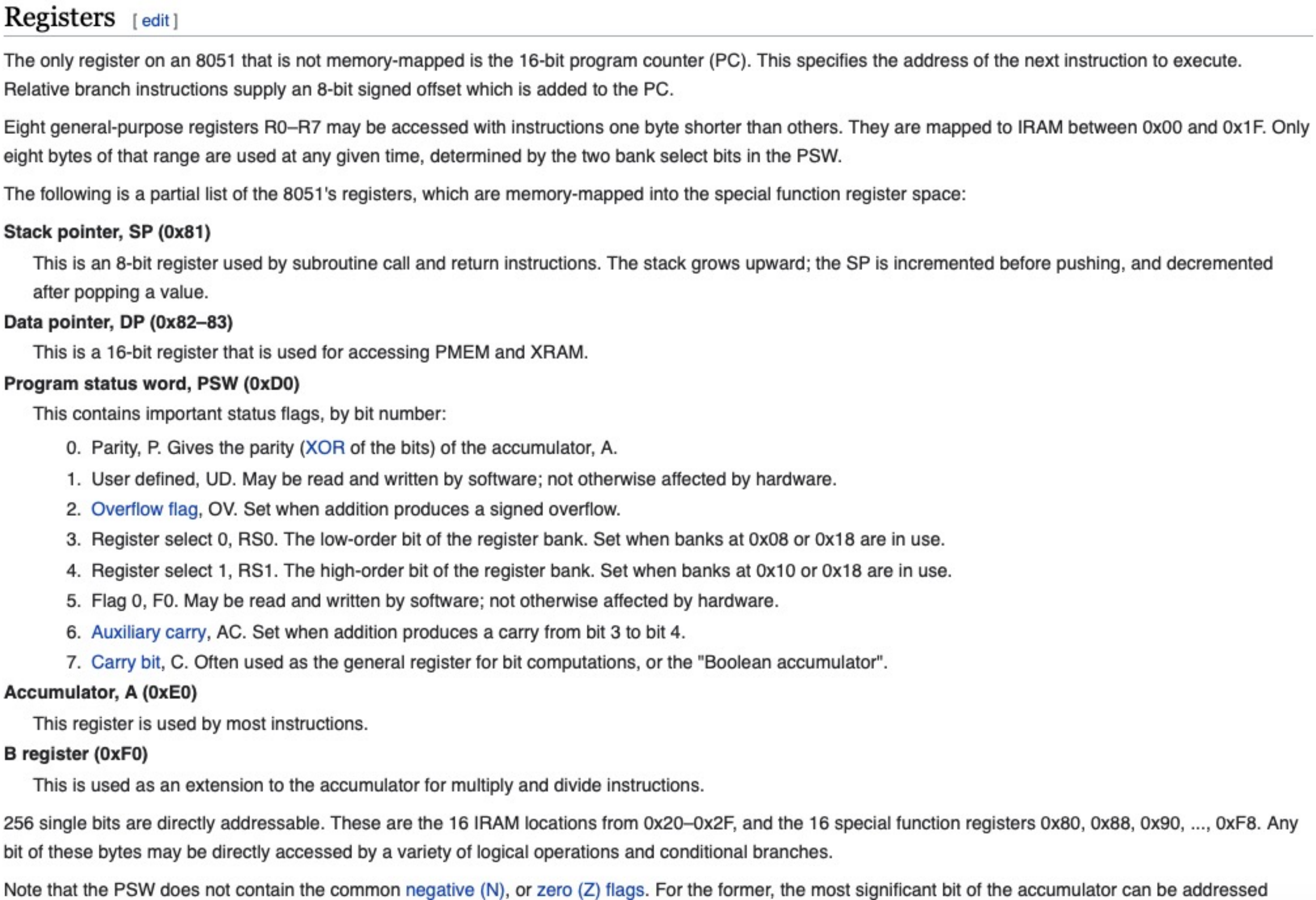
Special function registers (SFR) are located in the same address space as IRAM, at addresses 0x80 to 0xFF, and are accessed directly using the same instructions as for the lower half of IRAM. They cannot be accessed indirectly via @R0 or @R1; indirect access to those addresses will access the second half of IRAM.

Sixteen of the SFRs (those whose addresses are multiples of 8) are also bit-addressable.

Program memory [edit]

Program memory (PMEM, though less common in usage than IRAM and XRAM) is up to 64 KB of read-only memory, starting at address 0 in a separate address space. It may be on- or off-chip, depending on the particular model of chip being used. Program memory is read-only, though some variants of the 8051 use on-chip flash memory and provide a method of re-programming the memory in-system or in-application.

i8051 MCU



Registers [edit]

The only register on an 8051 that is not memory-mapped is the 16-bit program counter (PC). This specifies the address of the next instruction to execute. Relative branch instructions supply an 8-bit signed offset which is added to the PC.

Eight general-purpose registers R0–R7 may be accessed with instructions one byte shorter than others. They are mapped to IRAM between 0x00 and 0x1F. Only eight bytes of that range are used at any given time, determined by the two bank select bits in the PSW.

The following is a partial list of the 8051's registers, which are memory-mapped into the special function register space:

Stack pointer, SP (0x81)

This is an 8-bit register used by subroutine call and return instructions. The stack grows upward; the SP is incremented before pushing, and decremented after popping a value.

Data pointer, DP (0x82–83)

This is a 16-bit register that is used for accessing PMEM and XRAM.

Program status word, PSW (0xD0)

This contains important status flags, by bit number:

0. Parity, P. Gives the parity (XOR of the bits) of the accumulator, A.
1. User defined, UD. May be read and written by software; not otherwise affected by hardware.
2. Overflow flag, OV. Set when addition produces a signed overflow.
3. Register select 0, RS0. The low-order bit of the register bank. Set when banks at 0x08 or 0x18 are in use.
4. Register select 1, RS1. The high-order bit of the register bank. Set when banks at 0x10 or 0x18 are in use.
5. Flag 0, F0. May be read and written by software; not otherwise affected by hardware.
6. Auxiliary carry, AC. Set when addition produces a carry from bit 3 to bit 4.
7. Carry bit, C. Often used as the general register for bit computations, or the "Boolean accumulator".

Accumulator, A (0xE0)

This register is used by most instructions.

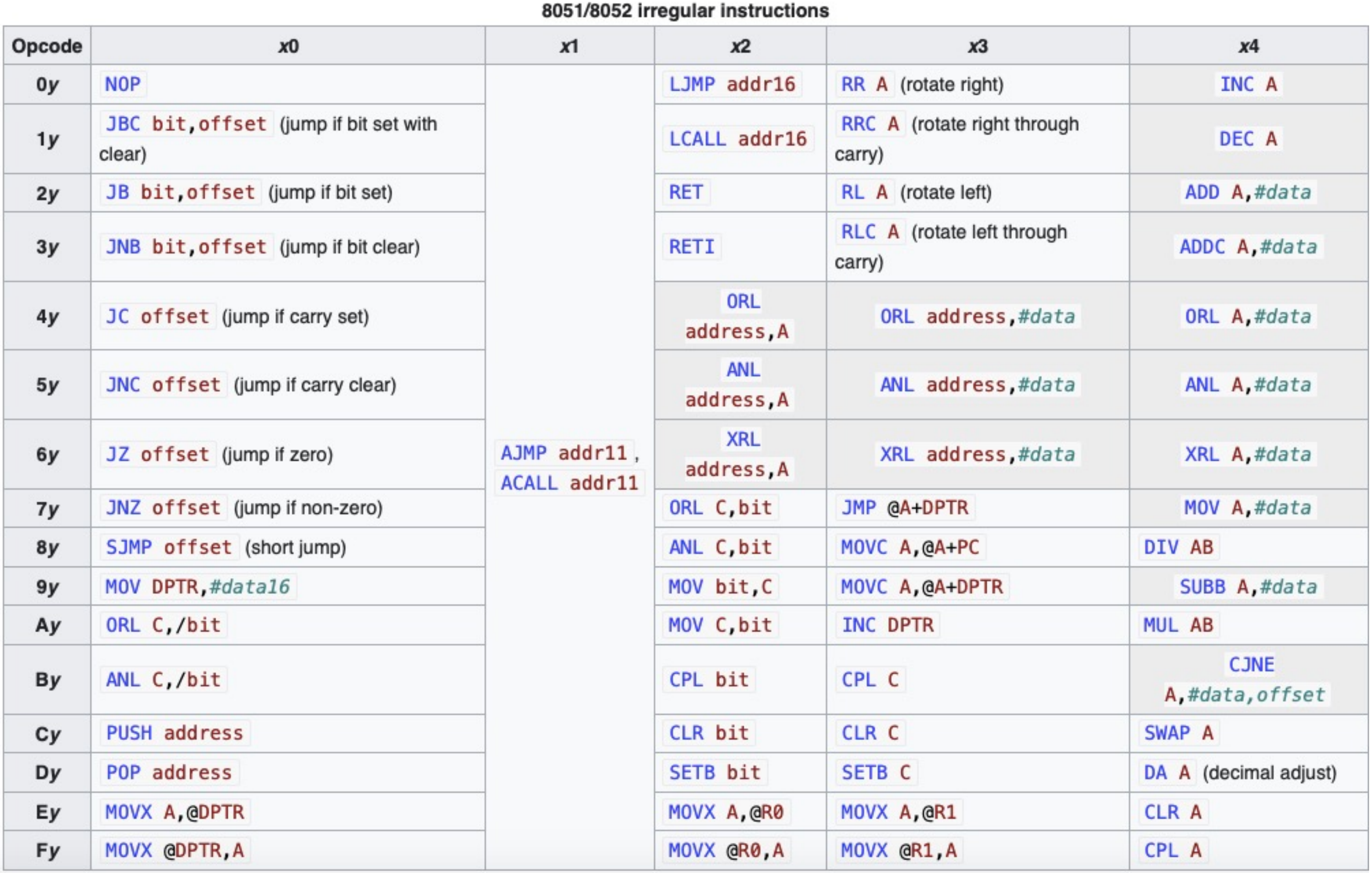
B register (0xF0)

This is used as an extension to the accumulator for multiply and divide instructions.

256 single bits are directly addressable. These are the 16 IRAM locations from 0x20–0x2F, and the 16 special function registers 0x80, 0x88, 0x90, ..., 0xF8. Any bit of these bytes may be directly accessed by a variety of logical operations and conditional branches.

Note that the PSW does not contain the common negative (N), or zero (Z) flags. For the former, the most significant bit of the accumulator can be addressed

i8051 MCU

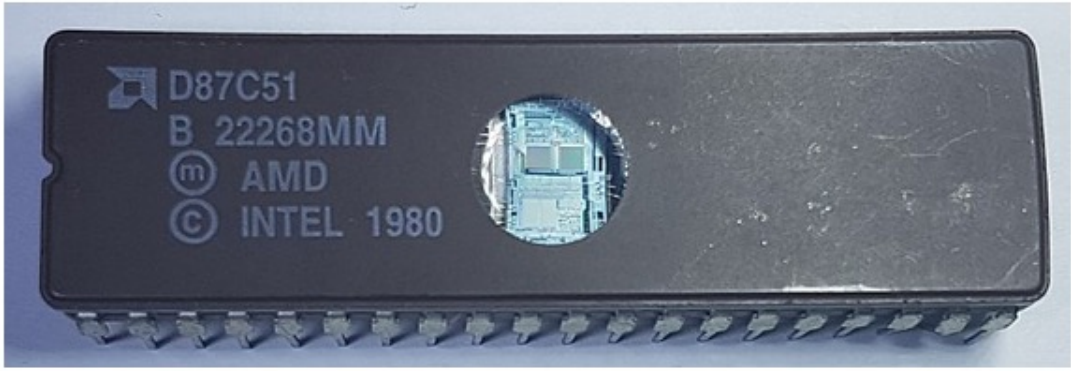


8051/8052 irregular instructions

Opcode	x0	x1	x2	x3	x4
0y	NOP	AJMP addr11, ACALL addr11	LJMP addr16	RR A (rotate right)	INC A
1y	JBC bit,offset (jump if bit set with clear)		LCALL addr16	RRC A (rotate right through carry)	DEC A
2y	JB bit,offset (jump if bit set)		RET	RL A (rotate left)	ADD A,#data
3y	JNB bit,offset (jump if bit clear)		RETI	RLC A (rotate left through carry)	ADDC A,#data
4y	JC offset (jump if carry set)		ORL address,A	ORL address,#data	ORL A,#data
5y	JNC offset (jump if carry clear)		ANL address,A	ANL address,#data	ANL A,#data
6y	JZ offset (jump if zero)		XRL address,A	XRL address,#data	XRL A,#data
7y	JNZ offset (jump if non-zero)		ORL C,bit	JMP @A+DPTR	MOV A,#data
8y	SJMP offset (short jump)		ANL C,bit	MOVC A,@A+PC	DIV AB
9y	MOV DPTR,#data16		MOV bit,C	MOVC A,@A+DPTR	SUBB A,#data
Ay	ORL C,/bit		MOV C,bit	INC DPTR	MUL AB
By	ANL C,/bit		CPL bit	CPL C	CJNE A,#data,offset
Cy	PUSH address		CLR bit	CLR C	SWAP A
Dy	POP address		SETB bit	SETB C	DA A (decimal adjust)
Ey	MOVX A,@DPTR		MOVX A,@R0	MOVX A,@R1	CLR A
Fy	MOVX @DPTR,A		MOVX @R0,A	MOVX @R1,A	CPL A

i8051 MCU

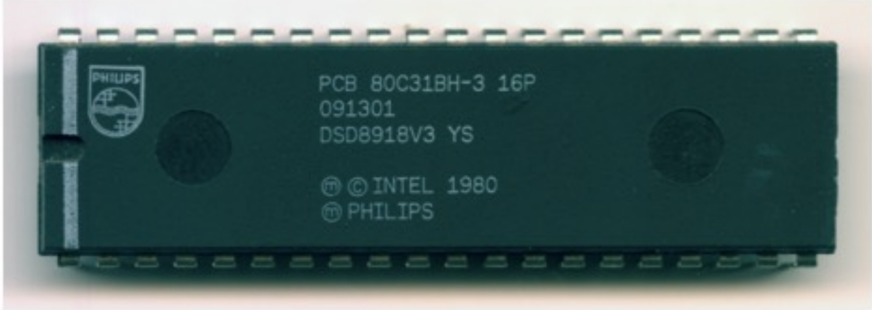
Intel MCS-51 second sources



AMD D87C51

MHS S-80C31

OKI M80C31

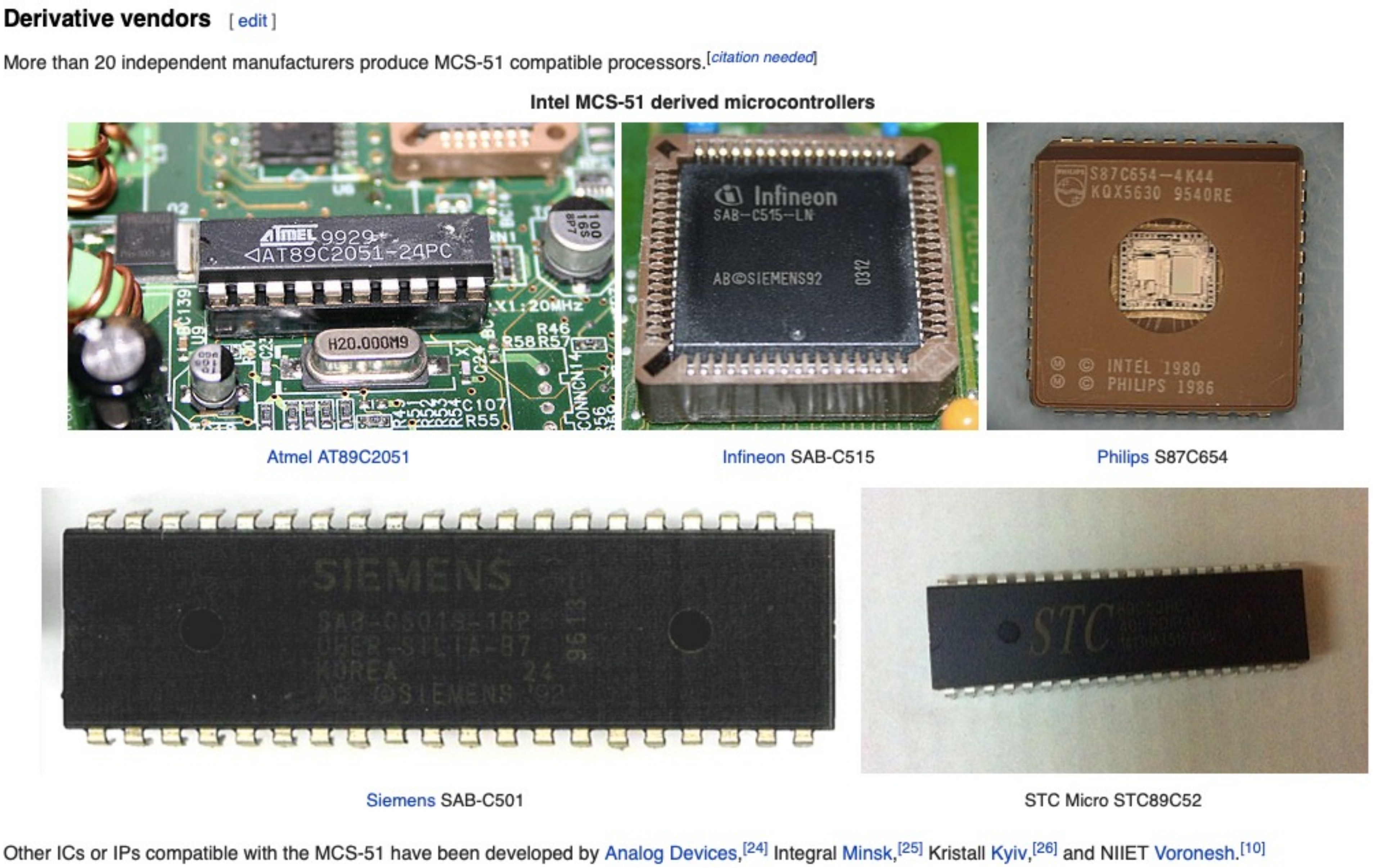


Philips PCB80C31

Signetics SCN8031

Temic TS80C32

i8051 MCU



Derivative vendors [edit]

More than 20 independent manufacturers produce MCS-51 compatible processors.[citation needed]

Intel MCS-51 derived microcontrollers

Atmel AT89C2051

Infineon SAB-C515

Philips S87C654

Siemens SAB-C501

STC Micro STC89C52

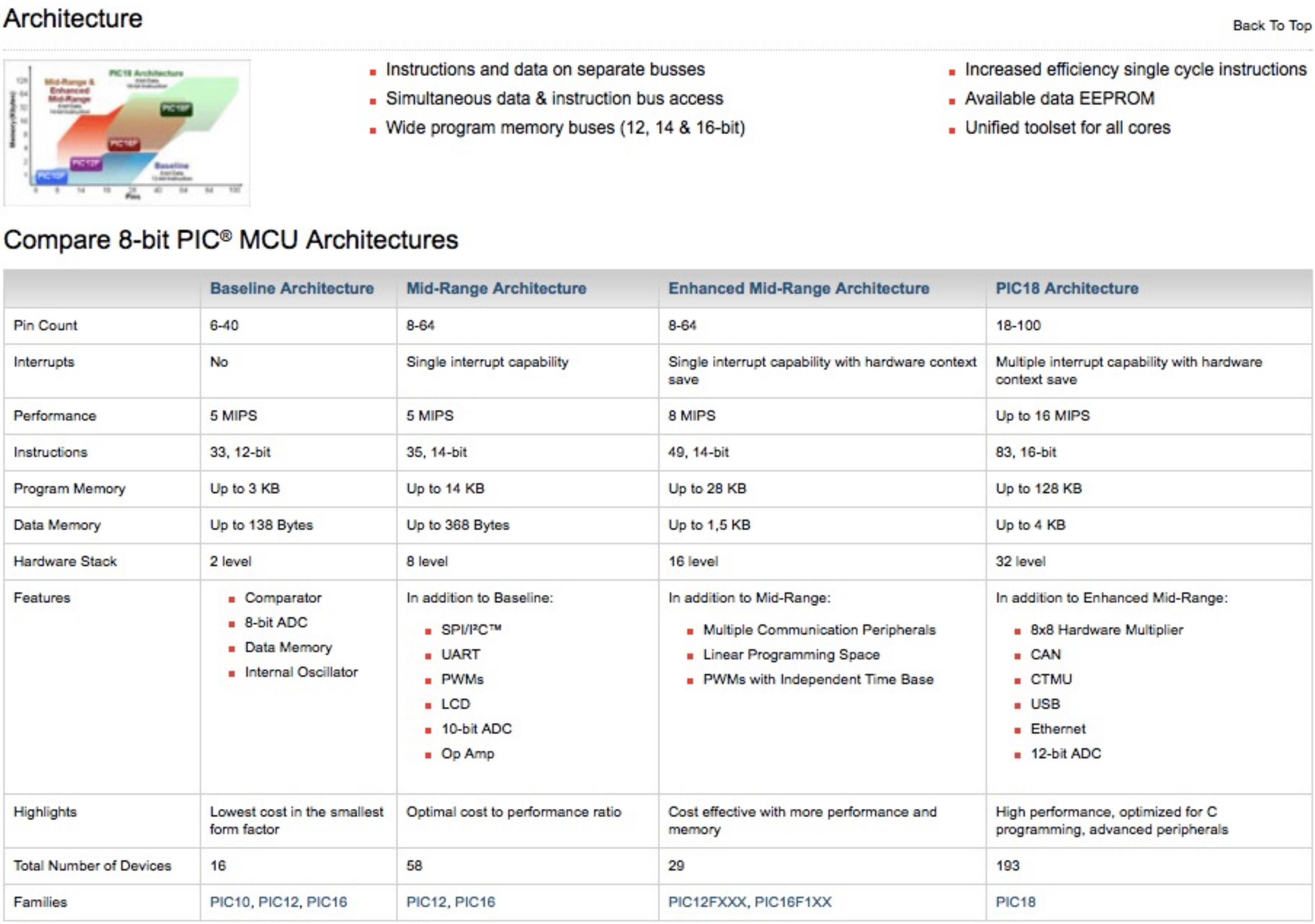
Other ICs or IPs compatible with the MCS-51 have been developed by Analog Devices,[24] Integral Minsk,[25] Kristall Kyiv,[26] and NIIET Voronesh.[10]

ISA

PIC

❑ PIC18F

PIC Families



Architecture

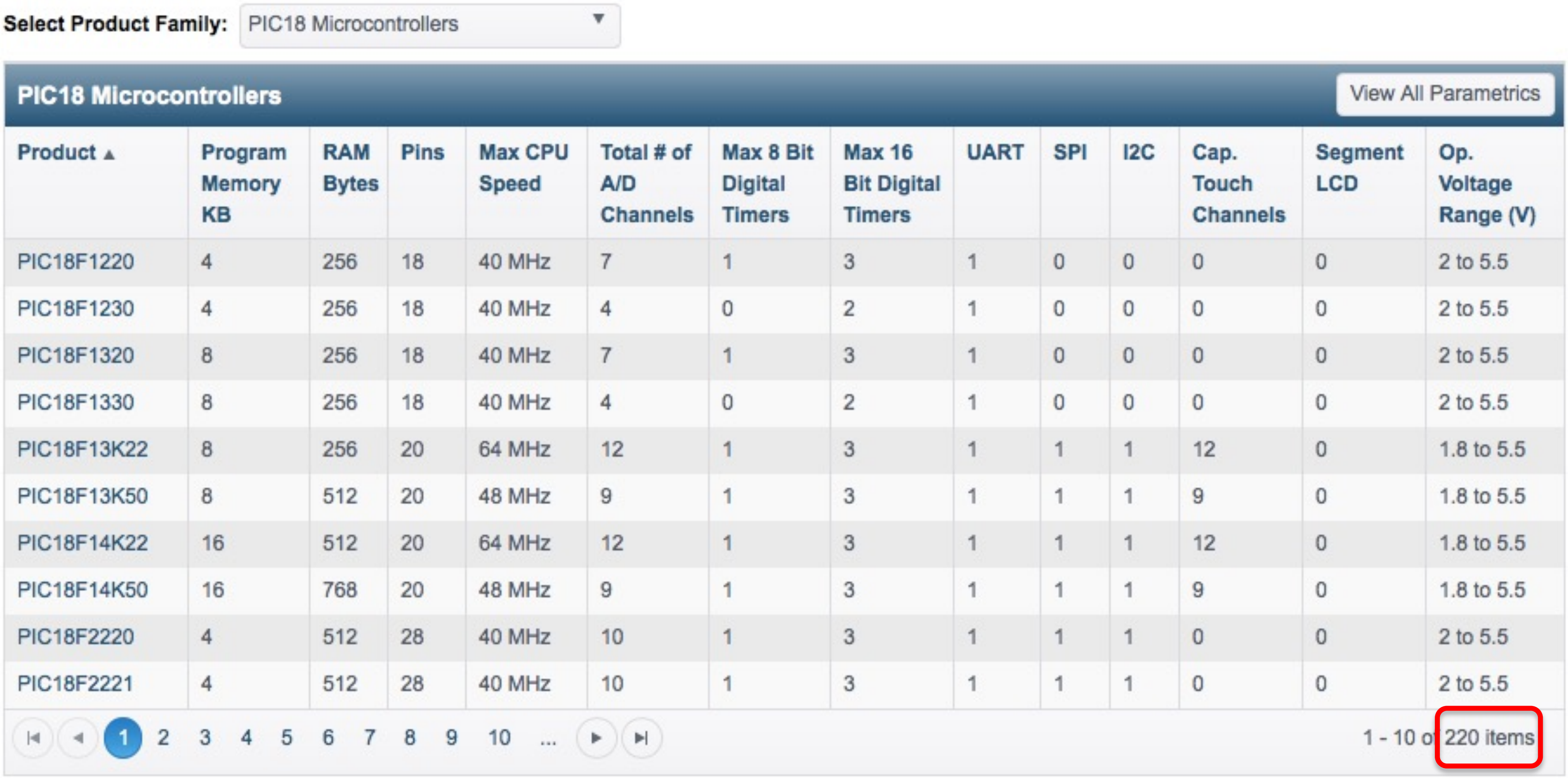
Back To Top

- Instructions and data on separate busses
- Simultaneous data & instruction bus access
- Wide program memory buses (12, 14 & 16-bit)
- Increased efficiency single cycle instructions
- Available data EEPROM
- Unified toolset for all cores

Compare 8-bit PIC® MCU Architectures

	Baseline Architecture	Mid-Range Architecture	Enhanced Mid-Range Architecture	PIC18 Architecture
Pin Count	6-40	8-64	8-64	18-100
Interrupts	No	Single interrupt capability	Single interrupt capability with hardware context save	Multiple interrupt capability with hardware context save
Performance	5 MIPS	5 MIPS	8 MIPS	Up to 16 MIPS
Instructions	33, 12-bit	35, 14-bit	49, 14-bit	83, 16-bit
Program Memory	Up to 3 KB	Up to 14 KB	Up to 28 KB	Up to 128 KB
Data Memory	Up to 138 Bytes	Up to 368 Bytes	Up to 1,5 KB	Up to 4 KB
Hardware Stack	2 level	8 level	16 level	32 level
Features	▪ Comparator ▪ 8-bit ADC ▪ Data Memory ▪ Internal Oscillator	In addition to Baseline: ▪ SPI/I²C™ ▪ UART ▪ PWMs ▪ LCD ▪ 10-bit ADC ▪ Op Amp	In addition to Mid-Range: ▪ Multiple Communication Peripherals ▪ Linear Programming Space ▪ PWMs with Independent Time Base	In addition to Enhanced Mid-Range: ▪ 8x8 Hardware Multiplier ▪ CAN ▪ CTMU ▪ USB ▪ Ethernet ▪ 12-bit ADC
Highlights	Lowest cost in the smallest form factor	Optimal cost to performance ratio	Cost effective with more performance and memory	High performance, optimized for C programming, advanced peripherals
Total Number of Devices	16	58	29	193
Families	PIC10, PIC12, PIC16	PIC12, PIC16	PIC12FXXX, PIC16F1XX	PIC18

PIC18F Family



Select Product Family: PIC18 Microcontrollers ▼

PIC18 Microcontrollers

Product ▲	Program Memory KB	RAM Bytes	Pins	Max CPU Speed	Total # of A/D Channels	Max 8 Bit Digital Timers	Max 16 Bit Digital Timers	UART	SPI	I2C	Cap. Touch Channels	Segment LCD	Op. Voltage Range (V)
PIC18F1220	4	256	18	40 MHz	7	1	3	1	0	0	0	0	2 to 5.5
PIC18F1230	4	256	18	40 MHz	4	0	2	1	0	0	0	0	2 to 5.5
PIC18F1320	8	256	18	40 MHz	7	1	3	1	0	0	0	0	2 to 5.5
PIC18F1330	8	256	18	40 MHz	4	0	2	1	0	0	0	0	2 to 5.5
PIC18F13K22	8	256	20	64 MHz	12	1	3	1	1	1	12	0	1.8 to 5.5
PIC18F13K50	8	512	20	48 MHz	9	1	3	1	1	1	9	0	1.8 to 5.5
PIC18F14K22	16	512	20	64 MHz	12	1	3	1	1	1	12	0	1.8 to 5.5
PIC18F14K50	16	768	20	48 MHz	9	1	3	1	1	1	9	0	1.8 to 5.5
PIC18F2220	4	512	28	40 MHz	10	1	3	1	1	1	0	0	2 to 5.5
PIC18F2221	4	512	28	40 MHz	10	1	3	1	1	1	0	0	2 to 5.5

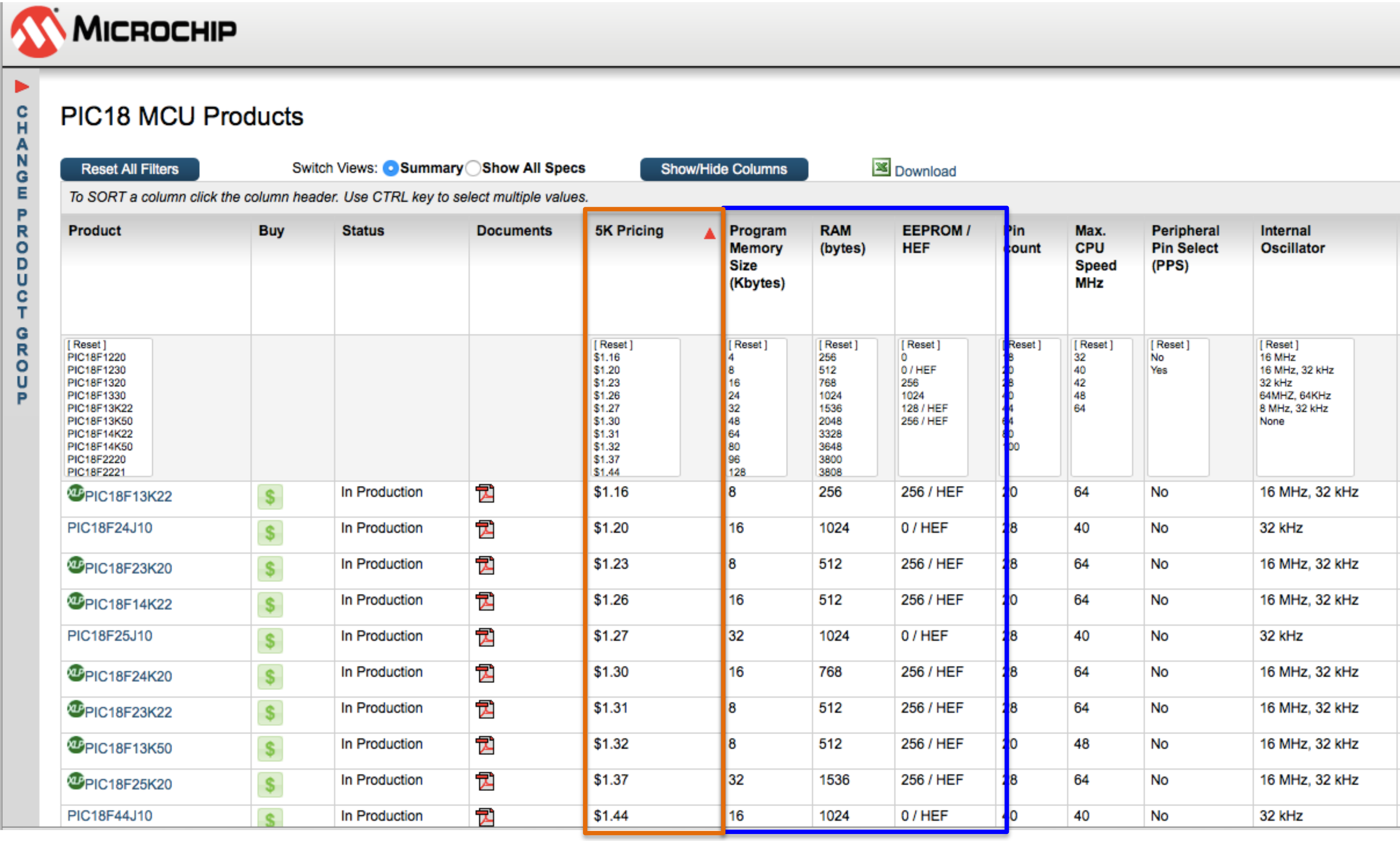
1 - 10 of 220 items

http://www.**microchip.com**/pagehandler/en-us/family/8bit/

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

PIC18F Products

DR JEFF SOFTWARE
INDIE APP DEVELOPER



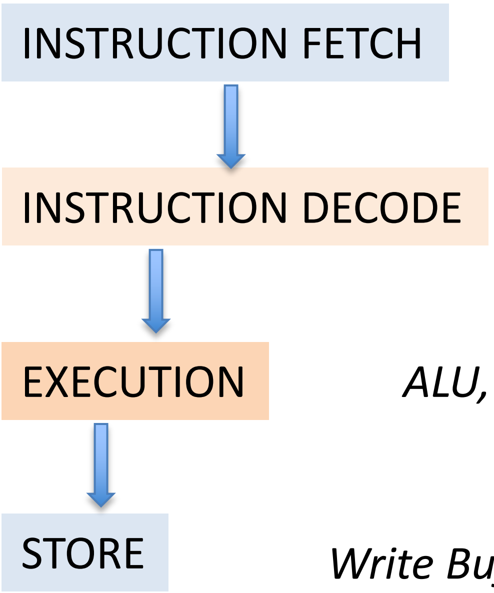
PIC18 MCU Products

To SORT a column click the column header. Use CTRL key to select multiple values.

Product	Buy	Status	Documents	5K Pricing ▲	Program Memory Size (Kbytes)	RAM (bytes)	EEPROM / HEF	Pin Count	Max. CPU Speed MHz	Peripheral Pin Select (PPS)	Internal Oscillator
[Reset] PIC18F1220 PIC18F1230 PIC18F1320 PIC18F1330 PIC18F13K22 PIC18F13K50 PIC18F14K22 PIC18F14K50 PIC18F2220 PIC18F2221				[Reset] $1.16 $1.20 $1.23 $1.26 $1.27 $1.30 $1.31 $1.32 $1.37 $1.44	[Reset] 4 8 16 24 32 48 64 80 96 128	[Reset] 256 512 768 1024 1536 2048 3328 3648 3800 3808	[Reset] 0 0 / HEF 256 1024 128 / HEF 256 / HEF	[Reset] 18 20 28 40 44 64 80 100	[Reset] 32 40 42 48 64	[Reset] No Yes	[Reset] 16 MHz 16 MHz, 32 kHz 32 kHz 64MHZ, 64KHz 8 MHz, 32 kHz None
XLP PIC18F13K22	$	In Production		$1.16	8	256	256 / HEF	20	64	No	16 MHz, 32 kHz
PIC18F24J10	$	In Production		$1.20	16	1024	0 / HEF	28	40	No	32 kHz
XLP PIC18F23K20	$	In Production		$1.23	8	512	256 / HEF	28	64	No	16 MHz, 32 kHz
XLP PIC18F14K22	$	In Production		$1.26	16	512	256 / HEF	20	64	No	16 MHz, 32 kHz
PIC18F25J10	$	In Production		$1.27	32	1024	0 / HEF	28	40	No	32 kHz
XLP PIC18F24K20	$	In Production		$1.30	16	768	256 / HEF	28	64	No	16 MHz, 32 kHz
XLP PIC18F23K22	$	In Production		$1.31	8	512	256 / HEF	28	64	No	16 MHz, 32 kHz
XLP PIC18F13K50	$	In Production		$1.32	8	512	256 / HEF	20	48	No	16 MHz, 32 kHz
XLP PIC18F25K20	$	In Production		$1.37	32	1536	256 / HEF	28	64	No	16 MHz, 32 kHz
PIC18F44J10	$	In Production		$1.44	16	1024	0 / HEF	40	40	No	32 kHz

Pipelining

PIC18F

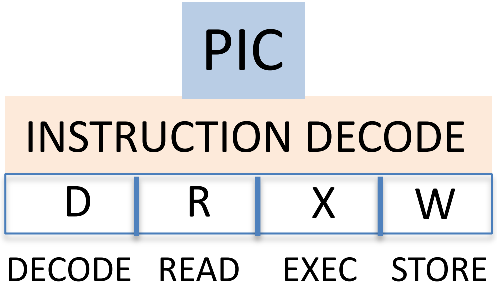


Program Memory Read

Opcode + Data operand fetch

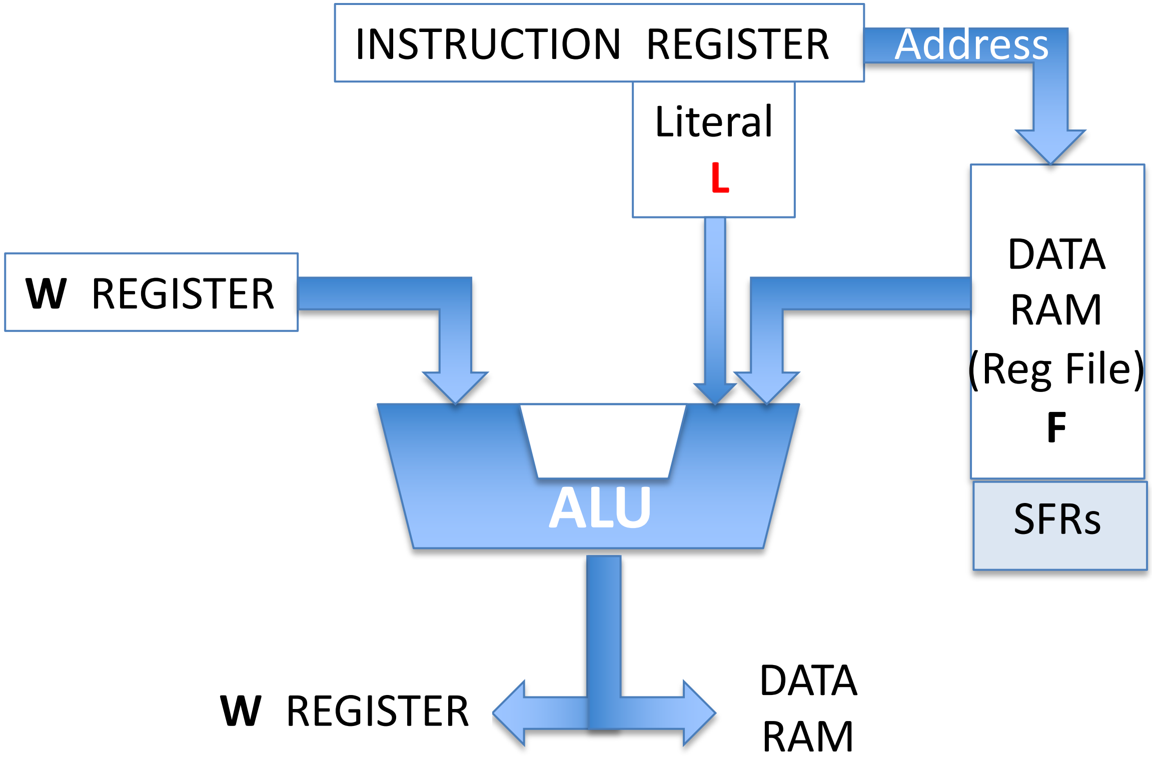
ALU, Move

Write Buffer -> Memory



Instruction cycle =
4x clock frequency

ALU Operands

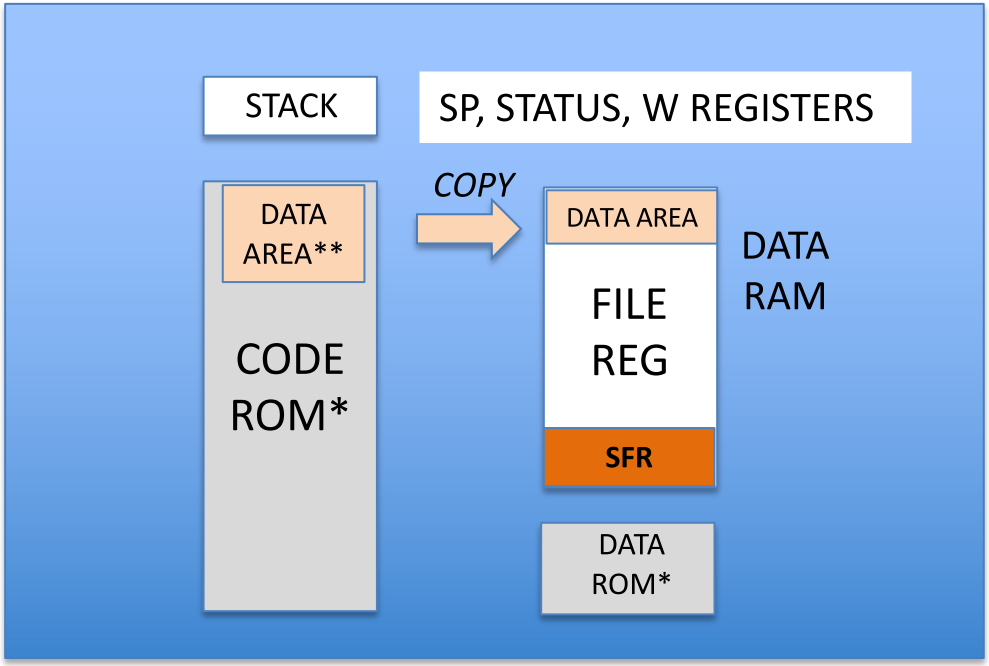


ADD **WF**: W+F → **W** -or- → **F**

ADD **LW**: L+W → **W**

PIC18F Dual Memory

PIC 18F MICROCONTROLLER



*READ ONLY
use **DB, DW

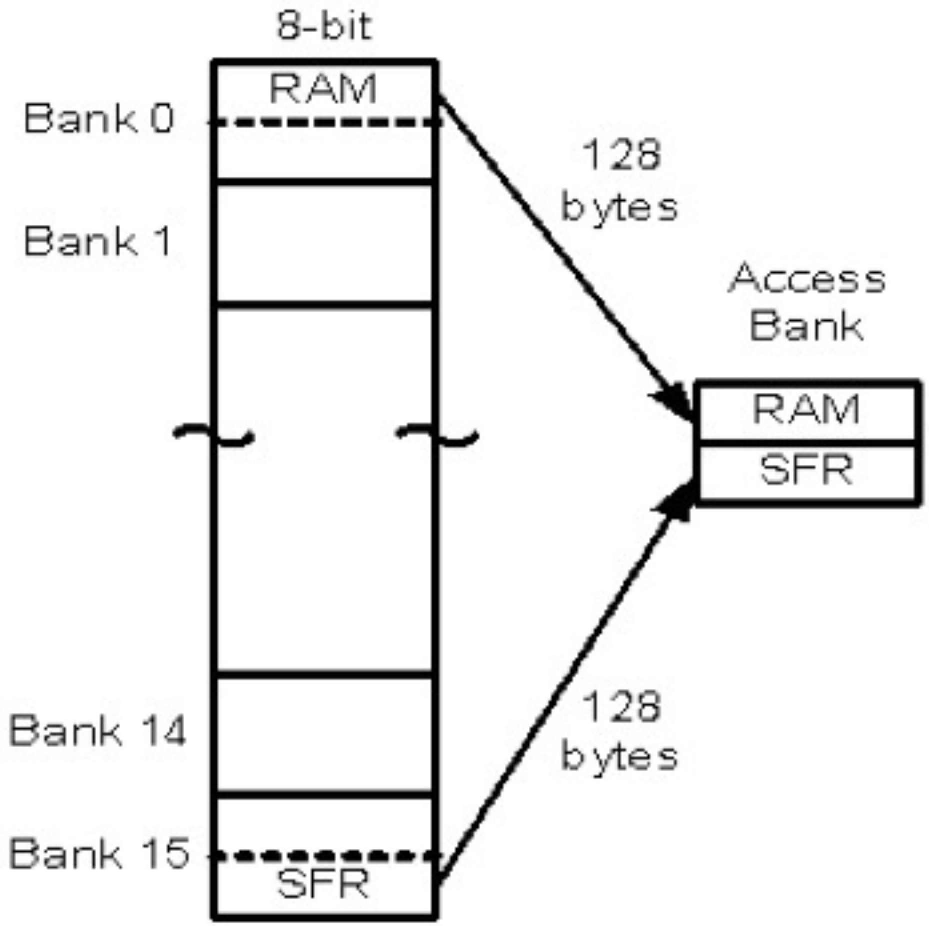
CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

RAM Banks

DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2022

PIC18F

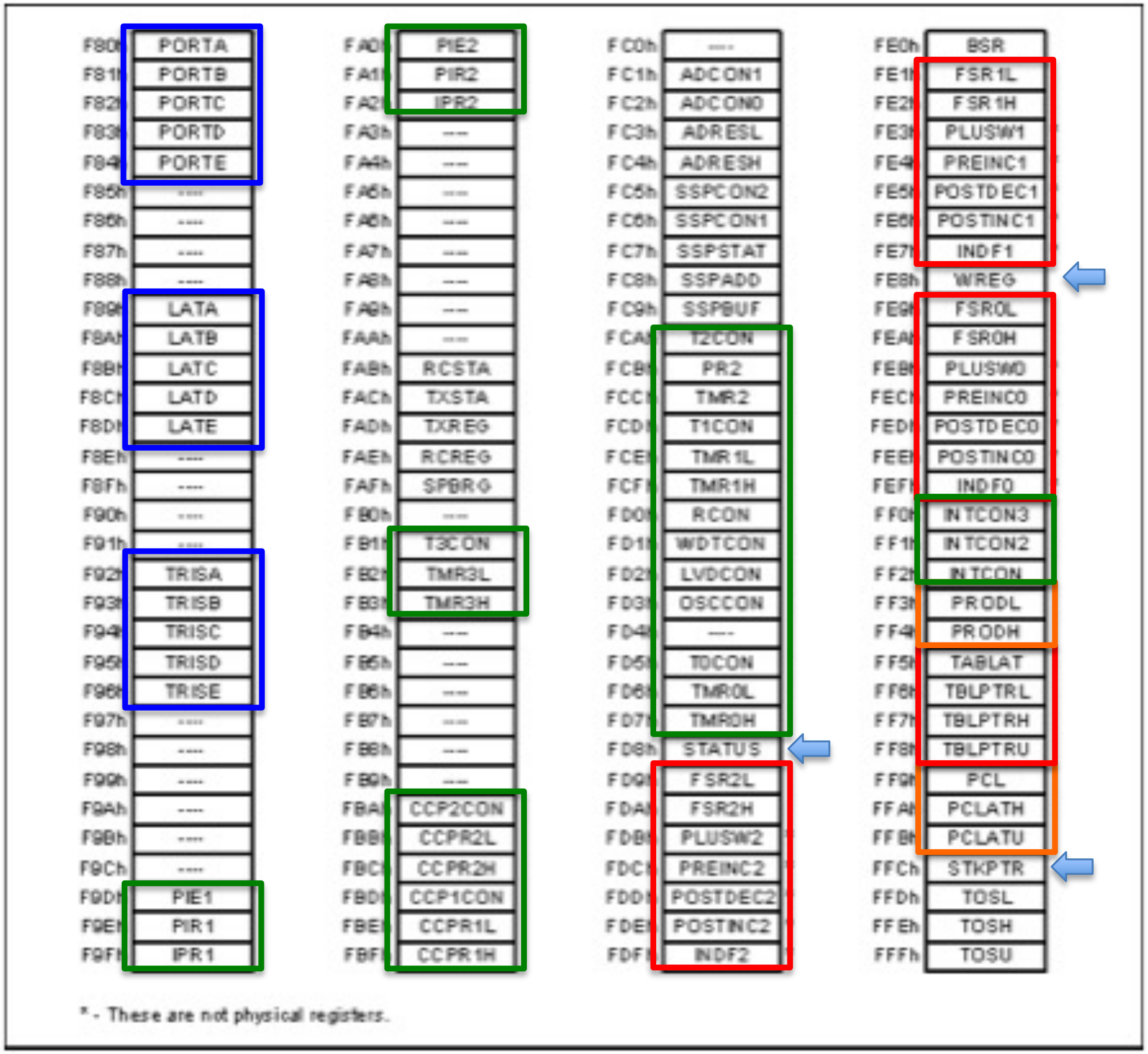
BSR (0-3)



SFRs

PIC18F

Figure 2-4. Special Function Registers of the PIC18 Family.



F80h	PORTA	FA0h	PIE2	FC0h	----	FE0h	BSR
F81h	PORTB	FA1h	PIR2	FC1h	ADCON1	FE1h	FSR1L
F82h	PORTC	FA2h	IPR2	FC2h	ADCON0	FE2h	FSR1H
F83h	PORTD	FA3h	----	FC3h	ADRESL	FE3h	PLUSW1
F84h	PORTE	FA4h	----	FC4h	ADRESH	FE4h	PREINC1
F85h	----	FA5h	----	FC5h	SSPCON2	FE5h	POSTDEC1
F86h	----	FA6h	----	FC6h	SSPCON1	FE6h	POSTINC1
F87h	----	FA7h	----	FC7h	SSPSTAT	FE7h	INDF1
F88h	----	FA8h	----	FC8h	SSPADD	FE8h	WREG
F89h	LATA	FA9h	----	FC9h	SSPBUF	FE9h	FSR0L
F8Ah	LATB	FAAh	----	FCAh	T2CON	FEAh	FSR0H
F8Bh	LATC	FABh	RCSTA	FCBh	PR2	FEBh	PLUSW0
F8Ch	LATD	FACh	TXSTA	FCCh	TMR2	FECh	PREINC0
F8Dh	LATE	FADh	TXREG	FCDh	T1CON	FEDh	POSTDEC0
F8Eh	----	FAEh	RCREG	FCEh	TMR1L	FEEh	POSTINC0
F8Fh	----	FAFh	SPBRG	FCFh	TMR1H	FEFh	INDF0
F90h	----	FB0h	----	FD0h	RCON	FF0h	INTCON3
F91h	----	FB1h	T3CON	FD1h	WDTCON	FF1h	INTCON2
F92h	TRISA	FB2h	TMR3L	FD2h	LVDCON	FF2h	INTCON
F93h	TRISB	FB3h	TMR3H	FD3h	OSCCON	FF3h	PRODL
F94h	TRISC	FB4h	----	FD4h	----	FF4h	PRODH
F95h	TRISD	FB5h	----	FD5h	T0CON	FF5h	TABLAT
F96h	TRISE	FB6h	----	FD6h	TMR0L	FF6h	TBLPTRL
F97h	----	FB7h	----	FD7h	TMR0H	FF7h	TBLPTRH
F98h	----	FB8h	----	FD8h	STATUS	FF8h	TBLPTRU
F99h	----	FB9h	----	FD9h	FSR2L	FF9h	PCL
F9Ah	----	FBAh	CCP2CON	FDAh	FSR2H	FFAh	PCLATH
F9Bh	----	FBBh	CCPR2L	FDBh	PLUSW2	FFBh	PCLATU
F9Ch	----	FBCh	CCPR2H	FDCh	PREINC2	FFCh	STKPTR
F9Dh	PIE1	FBDh	CCP1CON	FDDh	POSTDEC2	FFDh	TOSL
F9Eh	PIR1	FBEh	CCPR1L	FDEh	POSTINC2	FFEh	TOSH
F9Fh	IPR1	FBFh	CCPR1H	FDFh	INDF2	FFFh	TOSU

* - These are not physical registers.

ALU & MOVE

PIC18F

ADD

Sets all flags

ADDLW <data>;Add W to Literal <data> → **W** (only)

ADDWF(C) <addr>, **W/F** ;Add W to F (Data RAM) → **W or F**

MOVE

Sets NO flags (1 exception)*

MOV**LW** <data> ;*Load* **L**iteral <data> → **W**

MOV**F*** <addr>, **W/F** ;*Load* **F** at <addr> → **W or F** (*same* location; **sets N, Z**)

MOV**WF** <addr> ;*Store* **W** → **F** at <addr>

MOV**FF** <addr1>,<addr2> ;*Move* **F1** → **F2** in DataRAM (*different* location)

MULTIPLY

Sets NO flags

MULLW ;Multiply W by **L**iteral <data> → PROD [H,L]

MULWF ;Multiply W by **F** at <addr> → PROD [H,L]

MOV**FF** PRODL ,<addr> ;Store PRODL* → **F** at <addr>

MOV**FF** PRODH,<addr> ;Store PRODH* → **F** at <addr>

**SFR*

Addressing Modes

PIC18F

- Direct (in instruction) MOVF/WF <addr>[8]

JUMP long-addr [21], BRA offset [+-7]

- Immediate (Literal Data) MOV/ADDLW k [-128 to +127]

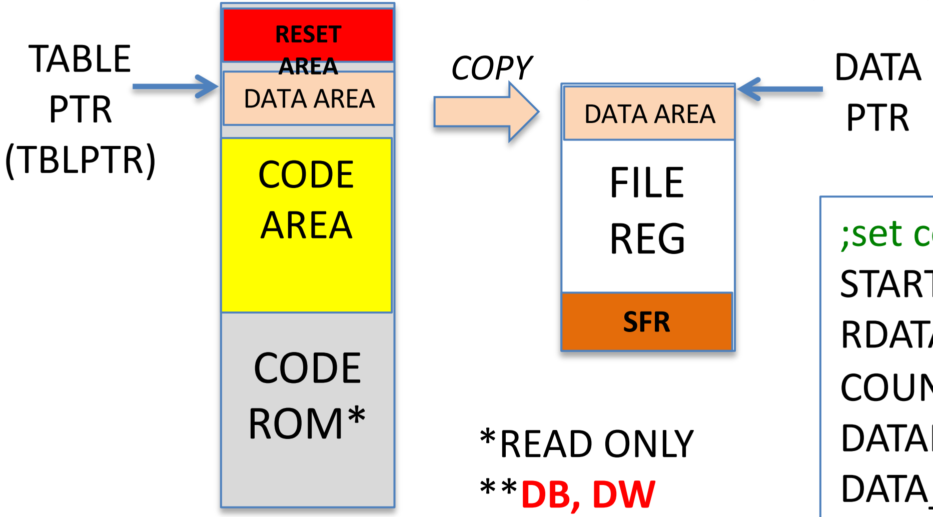
- Indirect (Register indirect, uses FSR)
  - **LFSR** n,<addr> [12-bit] (n=0, 1, 2)
  - MOVF/WF **INDF**n
  - CLRF/MOVF **POSTINC**n/**POSTDEC**n/**PREINC**n

- Indexed (Base Register FSR + Index Register W)
  - CLRF/MOVF **PLUSW**n
  - INCF <addr>,F ;increment F -or-
  - ADDLW 0x01 ;increment W

PIC18F Memory Map

Setup code

PIC 18F MICROCONTROLLER



*READ ONLY
**DB, DW

Include file (.inc)

```
;set code & data areas (EQU)
START   EQU   100H ;set code to start at 100H
RDATA_ORG   EQU   50H ;reserved 176 bytes for data
COUNTER   EQU   0 ;reserve counter at 0
DATAPTR   EQU   1 ;base of File Reg at 1
DATA_SIZE   EQU   D'25'

;create Data Table (DB) in CODE ROM
ZERO   DB   0
ONE   DB   1
TEN   DB   D'10'
ALL1   DB   FFH
…
;COPY Data Table from ROM to File Reg (next slide)
```

- Using **MPLab** IDE

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

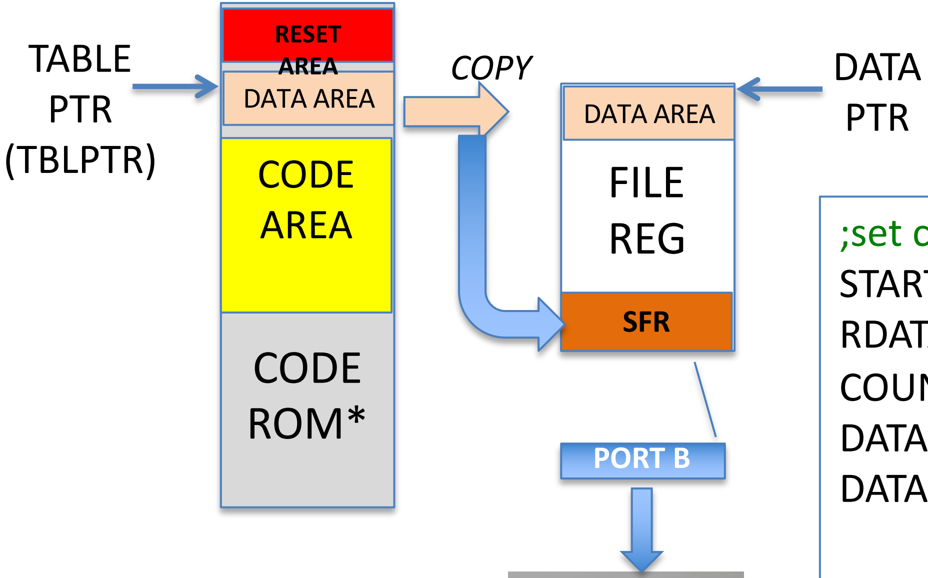
COMP122

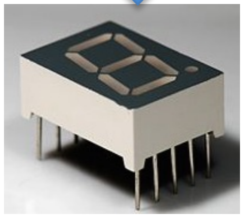
PIC18F Memory Map

DR JEFF SOFTWARE INDIE APP DEVELOPER
© Jeff Drobman
2016-2022

"HELLO WORLD"

PIC 18F MICROCONTROLLER





- Using **MPLab** IDE

Include file (.inc)

```
;set code & data areas (EQU)
START   EQU   100H ;set code to start at 100H
RDATA_ORG   EQU   50H ;reserved 176 bytes for data
COUNTER   EQU   0 ;reserve counter at 0
DATAPTR   EQU   1 ;base of File Reg at 1
DATA_SIZE   EQU   D'12'

;create Data Table (DB) in CODE ROM for message
HE   DW   A'HE'
LL   DW   A'LL'
OSP   DW   A'O '
WO   DW   A'WO'
RL   DW   A'RL'
D!   DW   A'D!'

;COPY Data Table from ROM to PORT B (next slide)
```

Headers

PIC18F

➢ Using **MPLab** IDE

```
;set jump to start (mandatory)
ORG   0
   GOTO    _start
; reserve code ROM areas for INTs
ORG   0x08  ;high priority Int
   ;output Int Ack or GOTO ISR
   RETFIE
ORG   0x18  ;low priority Int
   ;output Int Ack or GOTO ISR
   RETFIE
```

RESERVE CODE AREA

```
; declare useful constants – 'ORG' opt
ZERO   EQU   0
ONE    EQU   1
TEN    EQU   D'10' ;not just 10
ALL1   EQU   0xFF
START   EQU  0x100 ;address 256
```

DECLARE CONSTANTS

```
ZERO   DB  0
ONE    DB   1
TEN    DB   D'10'
ALL1   DB   0xFF
```

```
ORG   START
_start   NOP
; start your code here
```

START CODE

Use Headers

PIC18F

- Using **MPLab** IDE

```
#include   <p18Fnnnn.inc   ; "nnnn" = part number
#include  <constants.inc>
#include  <mylibrary.inc>

;my code starts here
<your code>

    SLEEP  ;or use an infinite loop
END
```

CREATE

Simulator issues

Copy Data Table

PIC18F

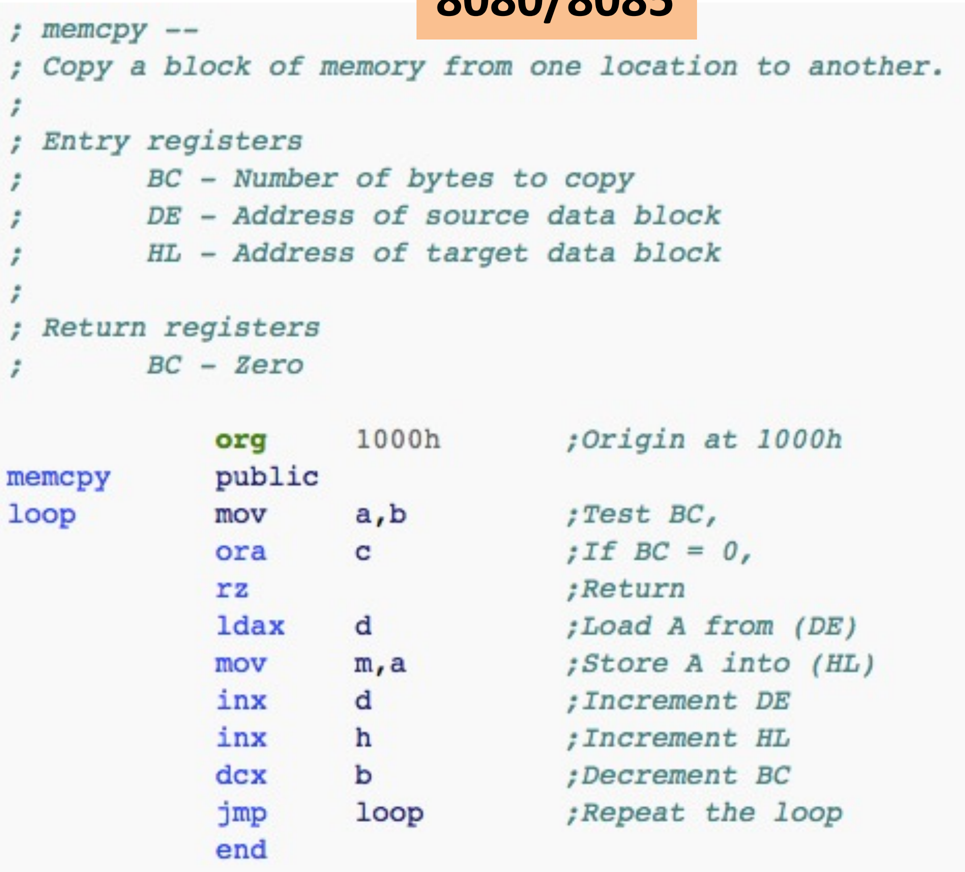
- Using **MPLab** IDE

```
        ;Init vars & pointer
        MOVLW   DATA_SIZE
        MOVWF   COUNTER ;init counter
        ;LFSR    1,DATAPTR – not needed

        ;init Table Pointer (21-bit address)
        CLRF   TBLPTRU ;upper 5 bits
        CLRF   TBLPTRH ;High byte
        MOVLW   RDATA_ORG
        MOVWF   TBLPTRL ; Low byte

        ;Copy ROM Table into Port B loop
LOOP    TBLRD  *+ ;read data from table into TABLAT & inc TBLPTR
        MOVFF  TABLAT,PORTB
        CALR    DELAY ;delay loop subroutine
        DECF    COUNTER
        BNZ     LOOP
```

DO THIS IN THE LAB

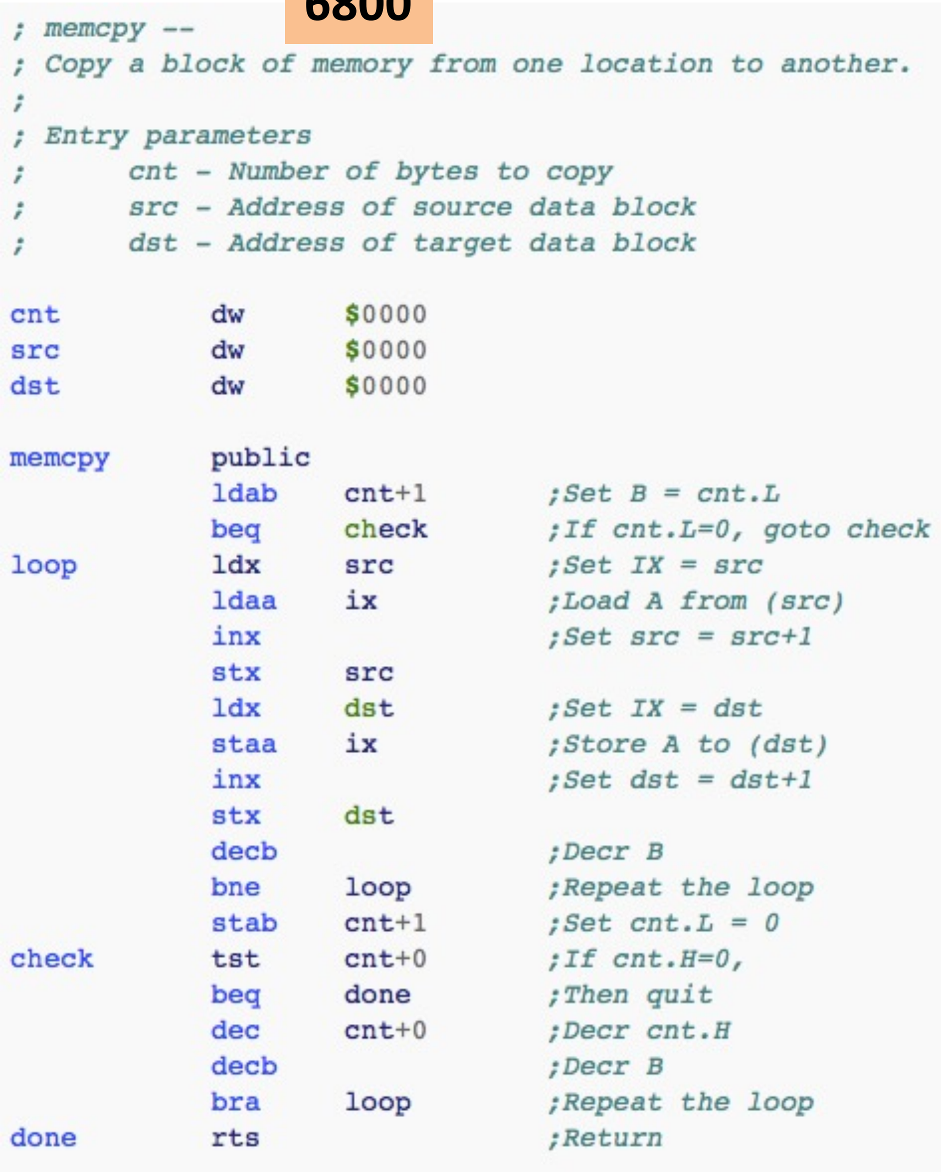
Copy Data Table

8080/8085



```
; memcpy --
; Copy a block of memory from one location to another.
;
; Entry registers
;       BC - Number of bytes to copy
;       DE - Address of source data block
;       HL - Address of target data block
;
; Return registers
;       BC - Zero

            org     1000h           ;Origin at 1000h
memcpy      public
loop        mov     a,b             ;Test BC,
            ora     c               ;If BC = 0,
            rz                      ;Return
            ldax    d               ;Load A from (DE)
            mov     m,a             ;Store A into (HL)
            inx     d               ;Increment DE
            inx     h               ;Increment HL
            dcx     b               ;Decrement BC
            jmp     loop            ;Repeat the loop
            end
```

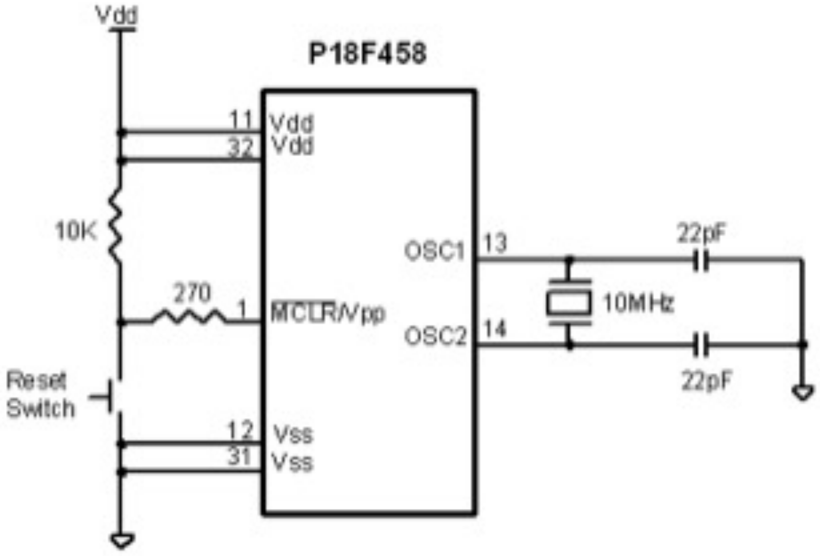
6800



```
; memcpy --
; Copy a block of memory from one location to another.
;
; Entry parameters
;       cnt - Number of bytes to copy
;       src - Address of source data block
;       dst - Address of target data block

cnt         dw      $0000
src         dw      $0000
dst         dw      $0000

memcpy      public
            ldab    cnt+1           ;Set B = cnt.L
            beq     check           ;If cnt.L=0, goto check
loop        ldx     src             ;Set IX = src
            ldaa    ix              ;Load A from (src)
            inx                     ;Set src = src+1
            stx     src
            ldx     dst             ;Set IX = dst
            staa    ix              ;Store A to (dst)
            inx                     ;Set dst = dst+1
            stx     dst
            decb                    ;Decr B
            bne     loop            ;Repeat the loop
            stab    cnt+1           ;Set cnt.L = 0
check       tst     cnt+0           ;If cnt.H=0,
            beq     done            ;Then quit
            dec     cnt+0           ;Decr cnt.H
            decb                    ;Decr B
            bra     loop            ;Repeat the loop
done        rts                     ;Return
```

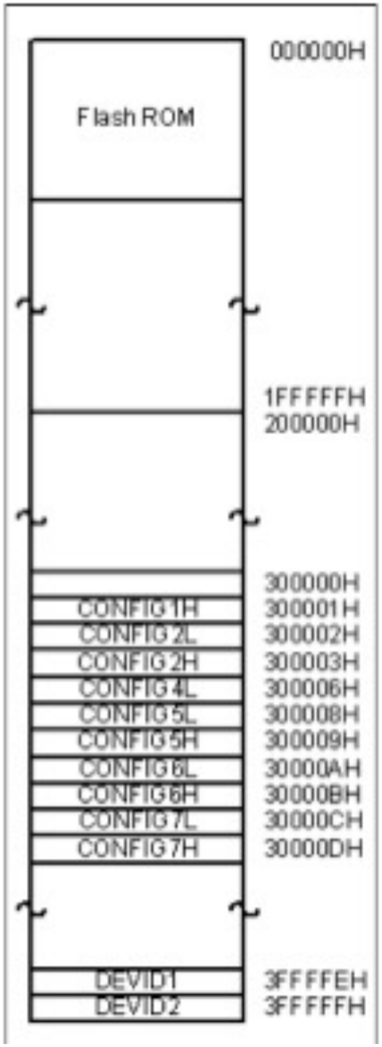
Connections Min

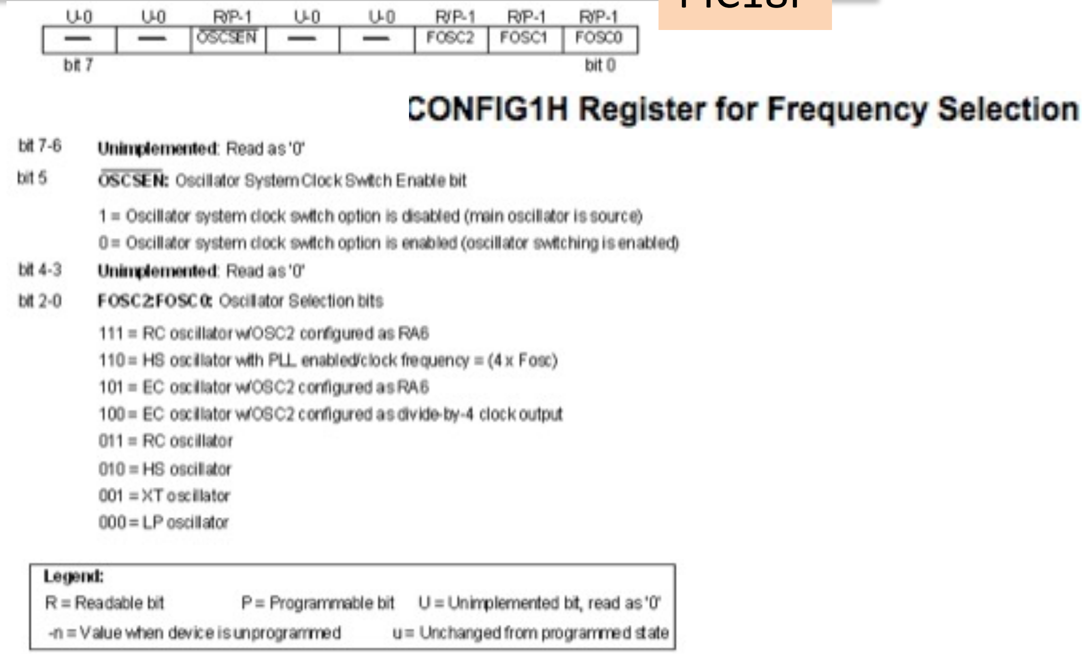
PIC18F



Configuration

PIC18F





U-0	U-0	R/P-1	U-0	U-0	R/P-1	R/P-1	R/P-1
—	—	OSCSEN	—	—	FOSC2	FOSC1	FOSC0
bit 7							bit 0

CONFIG1H Register for Frequency Selection

bit 7-6 **Unimplemented**: Read as '0'

bit 5 **OSCSEN**: Oscillator System Clock Switch Enable bit

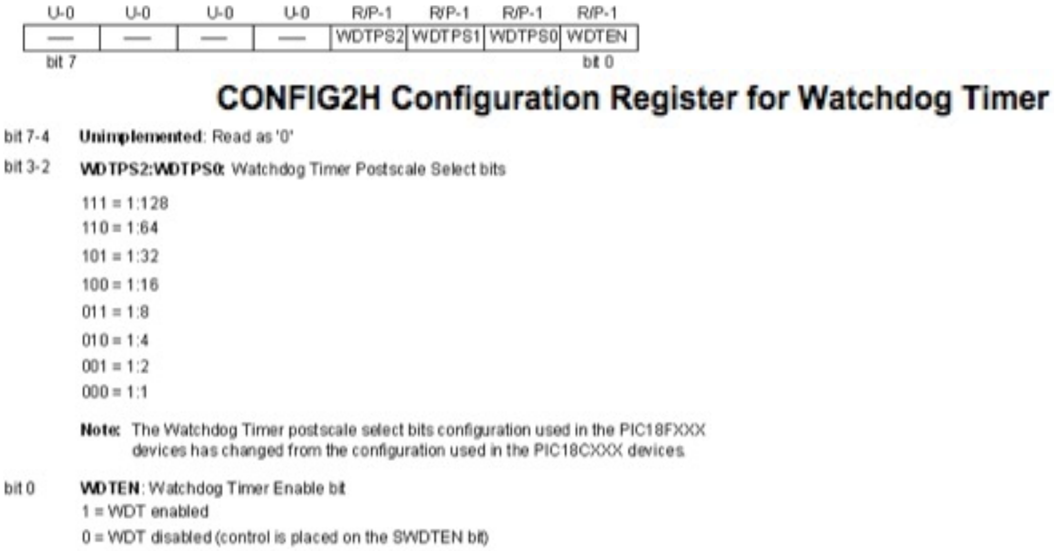
1 = Oscillator system clock switch option is disabled (main oscillator is source)
0 = Oscillator system clock switch option is enabled (oscillator switching is enabled)

bit 4-3 **Unimplemented**: Read as '0'

bit 2-0 **FOSC2:FOSC0**: Oscillator Selection bits

111 = RC oscillator w/OSC2 configured as RA6
110 = HS oscillator with PLL enabled/clock frequency = (4 x Fosc)
101 = EC oscillator w/OSC2 configured as RA6
100 = EC oscillator w/OSC2 configured as divide-by-4 clock output
011 = RC oscillator
010 = HS oscillator
001 = XT oscillator
000 = LP oscillator

Legend:
R = Readable bit P = Programmable bit U = Unimplemented bit, read as '0'
-n = Value when device is unprogrammed u = Unchanged from programmed state



U-0	U-0	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1
—	—	—	—	WDTPS2	WDTPS1	WDTPS0	WDTEN
bit 7							bit 0

CONFIG2H Configuration Register for Watchdog Timer

bit 7-4 **Unimplemented**: Read as '0'

bit 3-2 **WDTPS2:WDTPS0**: Watchdog Timer Postscale Select bits

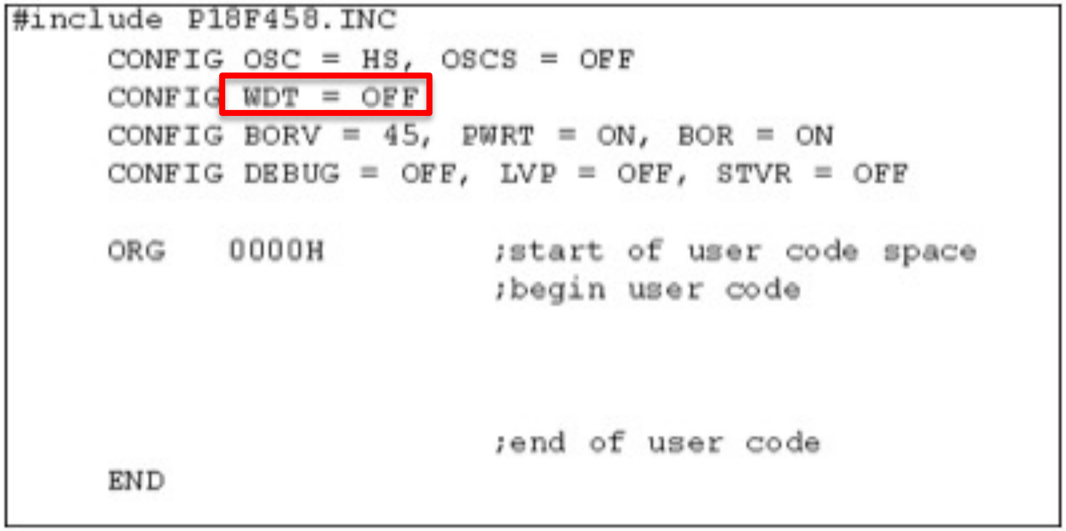
111 = 1:128
110 = 1:64
101 = 1:32
100 = 1:16
011 = 1:8
010 = 1:4
001 = 1:2
000 = 1:1

Note: The Watchdog Timer postscale select bits configuration used in the PIC18FXXX devices has changed from the configuration used in the PIC18CXXX devices.

bit 0 **WDTEN**: Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled (control is placed on the SWDTEN bit)

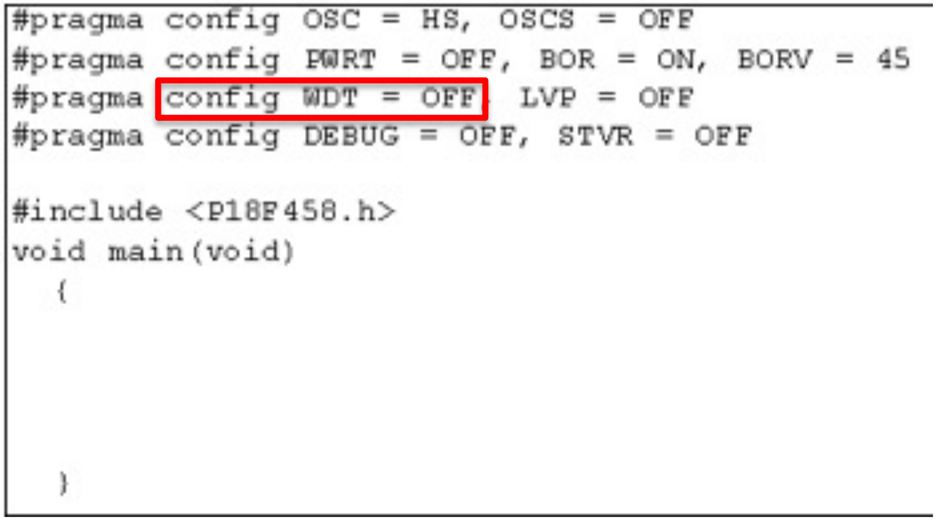
Config Code

PIC18F



```
#include P18F458.INC
     CONFIG OSC = HS, OSCS = OFF
     CONFIG WDT = OFF
     CONFIG BORV = 45, PWRT = ON, BOR = ON
     CONFIG DEBUG = OFF, LVP = OFF, STVR = OFF

     ORG    0000H           ;start of user code space
                            ;begin user code



                            ;end of user code
     END
```

ASSEMBLY



```
#pragma config OSC = HS, OSCS = OFF
#pragma config PWRT = OFF, BOR = ON, BORV = 45
#pragma config WDT = OFF, LVP = OFF
#pragma config DEBUG = OFF, STVR = OFF

#include <P18F458.h>
void main(void)
  {





  }
```

C

Interrupts

PIC18F

CLASSES

- MASKABLE
  - NMI (non)
  - INT (maskable)
- VECTORED
  - NVI (non – PIC)
  - VI
- PRIORITY (PIC)
  - High
  - Low (High INTs “preempt” Low)
- INTERNAL
  - Hardware events
    - Timers
    - ADC
    - I/O (S, P)
  - Software exceptions

PINS

- INT 0 → Pin 33
- INT 1 → Pin 34
- INT 2 → Pin 35

ENABLES

- GIE – global (all 3)
- INT 0-2
- PRIE – peripherals

PRIORITIES

- HIGH
- LOW

→ SAVED ON STACK

- **PC**
- W
- STATUS
- BSR

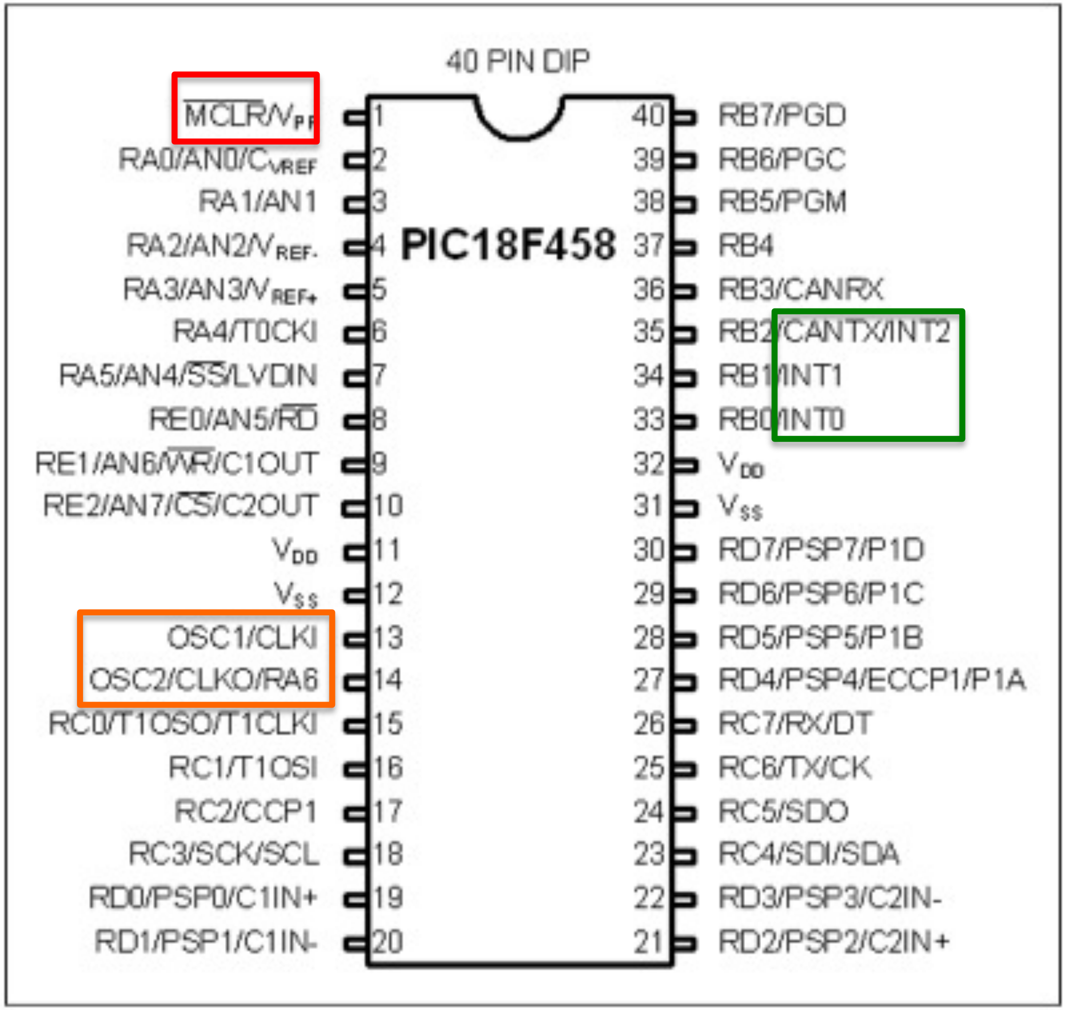
PROCESSOR STATE

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Interrupts/Clocks Pins

DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2022

PIC18F



Interrupts C Example

PIC18F

```
#include <p18f4321.h>
void    ISR (void);  //declares ISR as sub after main
#pragma code Int=0x08
void  Intasm( )
{
_asm   //use assembly code
GOTO ISR
_endasm
}
#pragma code  //org main
Void main(  )
   {
// do stuff here
While(1) {    }
}
#pragma interrupt ISR
Void ISR (void)     //interrupt svc routine
{   //do int stuff
   }
```

Config Interrupts in C

EXTERNAL INTS

PIC18F

```
Void   ISR( );  //declare the "ISR" subroutine
#pragma code int_vectH = 0x08  //assign int "vector" for High Pri Int
Void   IntH( )  {
_asm  //use assembly code here (no "GOTO" in C)
GOTO ISR
_endasm
#pragma code int_vectL = 0x18  //assign int "vector" for Low Pri Int
#pragma code  //main starts here (after the Ints)
Void   main ( )
{
ADCON1=0x0F;   //config Port B as input for interrupts
INTCONbits.INT0IE = 1  //enable INT0
INTCON3bits.INT1IE = 1  //enable INT1
INTCONbits.INT0F = 0  //clear flag
INTCON3bits.INT1F = 0 //clear flag
INTCON3bits.INT1IP = 0  //set INT1 to Low priority
RCONbits.IPEN = 1  //enable all priority interrupts
INTCONbits.GIEH = 1  //enable Global High priority interrupts
INTCONbits.GIEL = 1   //enable Global Low priority interrupts
While(1);   //wait for INT0 or INT1
}
```

SET ORGS

Or use a "Call":
ISR( )

SETUP

PINS

- INT 0 → RB0
- INT 1 → RB1
- INT 2 → RB2

Timers

PIC18F

Timers

- Timer 0
- Timer 1
- Timer 2
- Timer 3
- Watchdog

PRESCALER

- Clock pulse counter (a power of 2)
  - 2
  - 4
  - 8
  - 16
  - 32
  - 64
  - 128
  - 256

MODES

- Timers
  - Count UP or DOWN
  - Preset start value
  - INT on OVERFLOW
  - Use INTERNAL clock
- Counters
  - Use EXTERNAL clock (pin TnCKI)

Config Timers in C

PIC18F

Timers

- Timer 0
- Timer 1
- Timer 2
- Timer 3
- Watchdog

SETUP

```
#pragma code  //main starts here (after the Ints)
Void   main ( )
{
T0CON=0x06;   //config T0
TMR0H = 0XFF  //init T0 High byte
TMR0L = 0XF0  //init T0 Low byte =-16 (will count UP to 0)
INTCONbits.TMR0IF = 0  //clear Timer0 flag
```

RUN TIMER

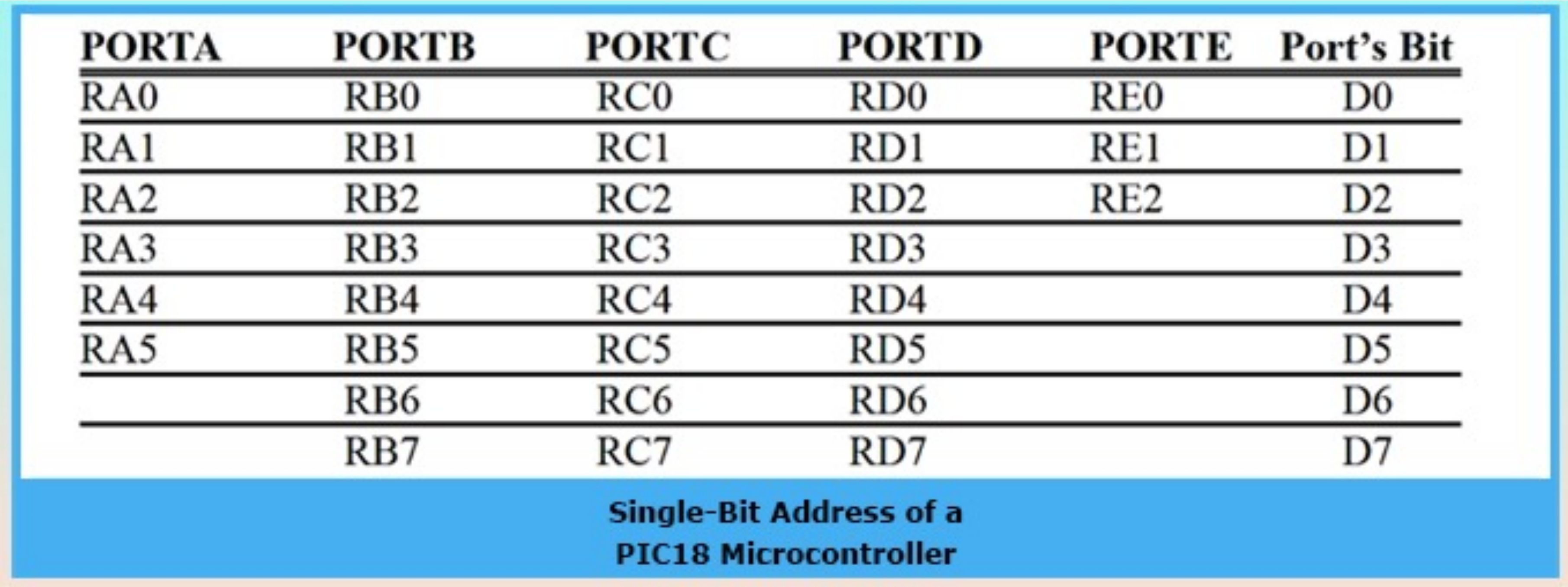
```
T0CONbits.TMR0ON = 1  //start timer
While(INTCONbits.TMR0IF ==0);  //wait for INT0 or INT1
T0CONbits.TMR0ON = 0  //stop timer
While(1);    //halt
}
```

Using Timer0 to create a **1ms delay**, assume:

- ❑ 8MHz crystal (=2MHz Instruction rate)
- ❑ Prescale value = 128 (2000/128=16)

Parallel Ports

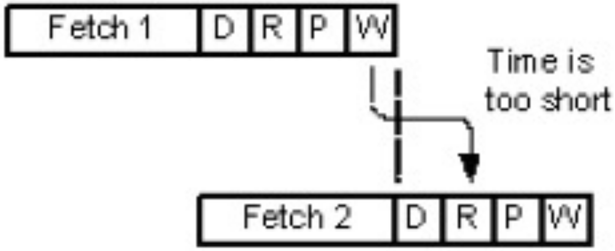
PIC18F



PORTA	PORTB	PORTC	PORTD	PORTE	Port's Bit
RA0	RB0	RC0	RD0	RE0	D0
RA1	RB1	RC1	RD1	RE1	D1
RA2	RB2	RC2	RD2	RE2	D2
RA3	RB3	RC3	RD3		D3
RA4	RB4	RC4	RD4		D4
RA5	RB5	RC5	RD5		D5
	RB6	RC6	RD6		D6
	RB7	RC7	RD7		D7

Single-Bit Address of a PIC18 Microcontroller

Write→Read

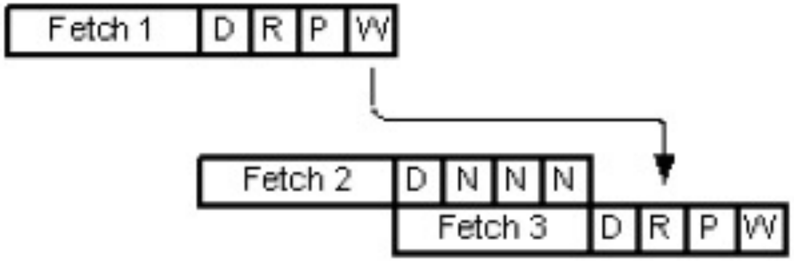


MOVF PORTC,W ;Read PORTC into WREG

MOVWF PORTB ;Write WREG to PORTB

The RAW (Read – After – Write) for two consecutive instructions.

INSTRUCTION



MOVF PORTC,W

NOP ;Bubble in Pipeline

MOVWF PORTB

Serial Ports

PIC18F

- USART – async or sync
- EIA (RS) 232 async
  - ❑ Framed: START, STOP bits
  - ❑ Handshake: RTS, CTS

COVERED

- SPI – sync
  - ❑ Clocked data (SCK)
  - ❑ SYNC characters
  - ❑ Master/slave (clock + data)
  - ❑ 2 registers
    - ▪ Tx/Rx shifter
    - ▪ Buffer

THIS WEEK

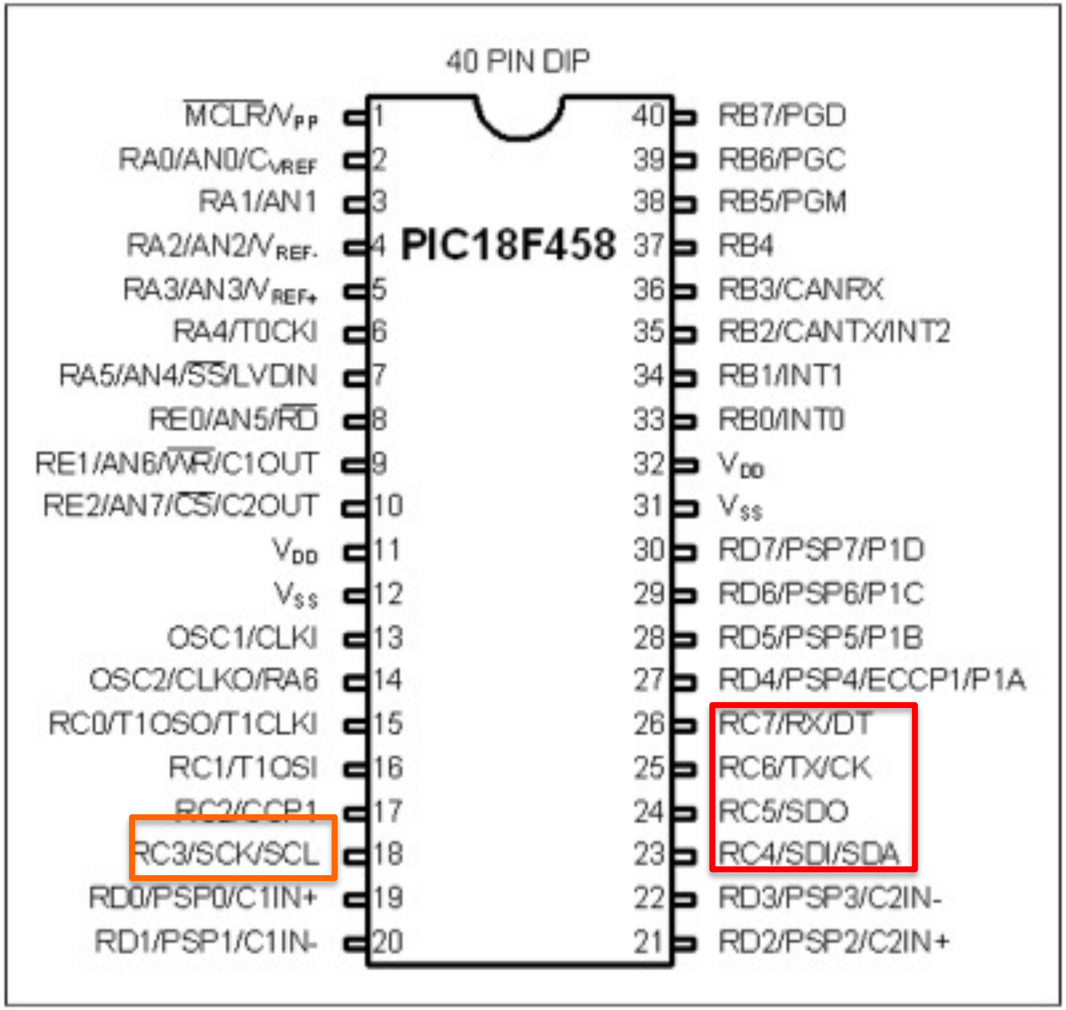
- I2C – sync
  - ❑ Peer to peer (IC to IC)

- SPI – sync (4 pins)
  - ▪ SDI
  - ▪ SDO
  - ▪ SDCLK

- I2C – sync (4 pins)
  - ▪ SDI
  - ▪ SDO
  - ▪ SDCLK

Serial Ports – Pins

PIC18F

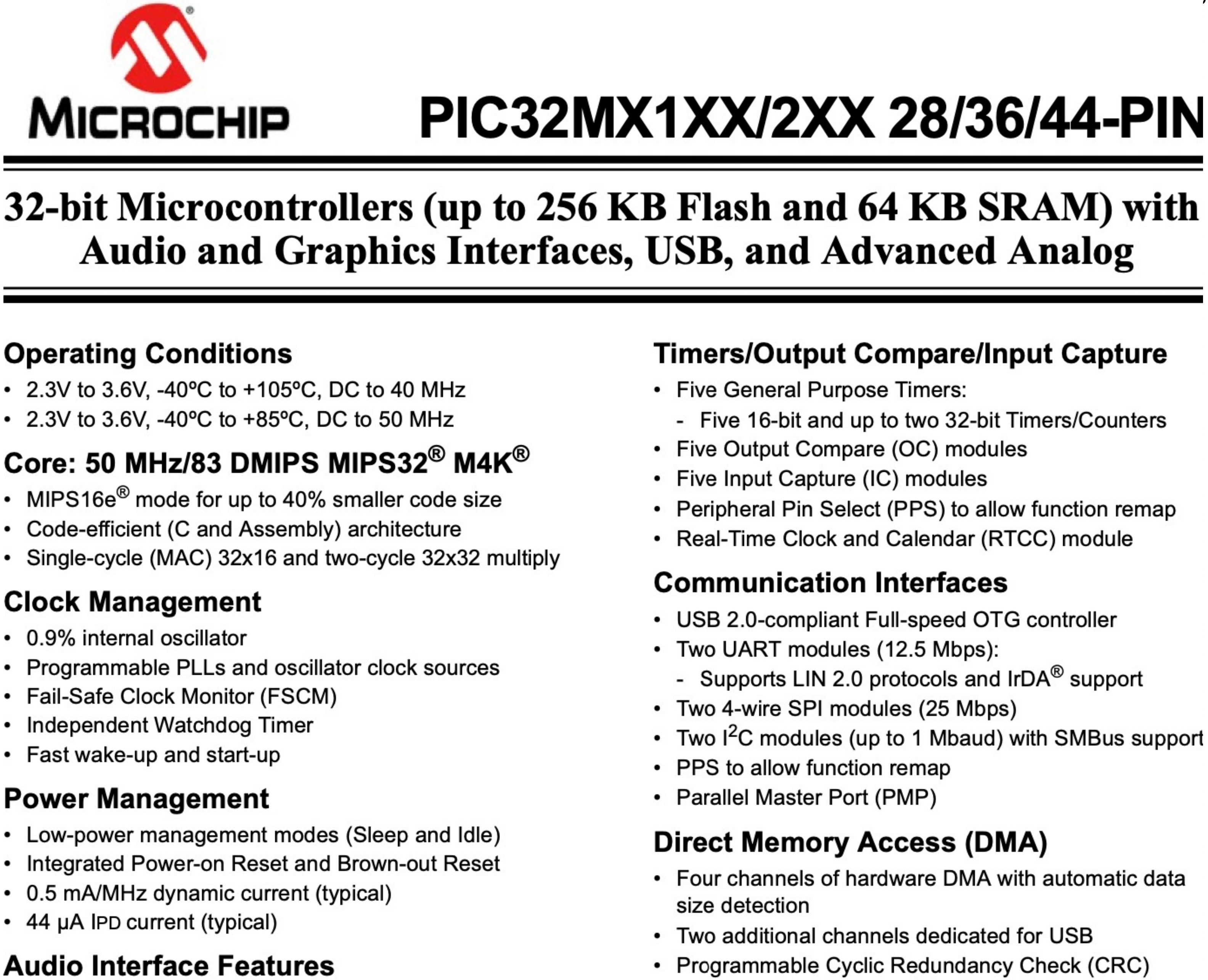


CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP122

PIC32 MCU

DR JEFF SOFTWARE
INDIE APP DEVELOPER



MICROCHIP

PIC32MX1XX/2XX 28/36/44-PIN

32-bit Microcontrollers (up to 256 KB Flash and 64 KB SRAM) with Audio and Graphics Interfaces, USB, and Advanced Analog

Operating Conditions

- 2.3V to 3.6V, -40ºC to +105ºC, DC to 40 MHz
- 2.3V to 3.6V, -40ºC to +85ºC, DC to 50 MHz

Core: 50 MHz/83 DMIPS MIPS32® M4K®

- MIPS16e® mode for up to 40% smaller code size
- Code-efficient (C and Assembly) architecture
- Single-cycle (MAC) 32x16 and two-cycle 32x32 multiply

Clock Management

- 0.9% internal oscillator
- Programmable PLLs and oscillator clock sources
- Fail-Safe Clock Monitor (FSCM)
- Independent Watchdog Timer
- Fast wake-up and start-up

Power Management

- Low-power management modes (Sleep and Idle)
- Integrated Power-on Reset and Brown-out Reset
- 0.5 mA/MHz dynamic current (typical)
- 44 µA IPD current (typical)

Audio Interface Features

Timers/Output Compare/Input Capture

- Five General Purpose Timers:
  - Five 16-bit and up to two 32-bit Timers/Counters
- Five Output Compare (OC) modules
- Five Input Capture (IC) modules
- Peripheral Pin Select (PPS) to allow function remap
- Real-Time Clock and Calendar (RTCC) module

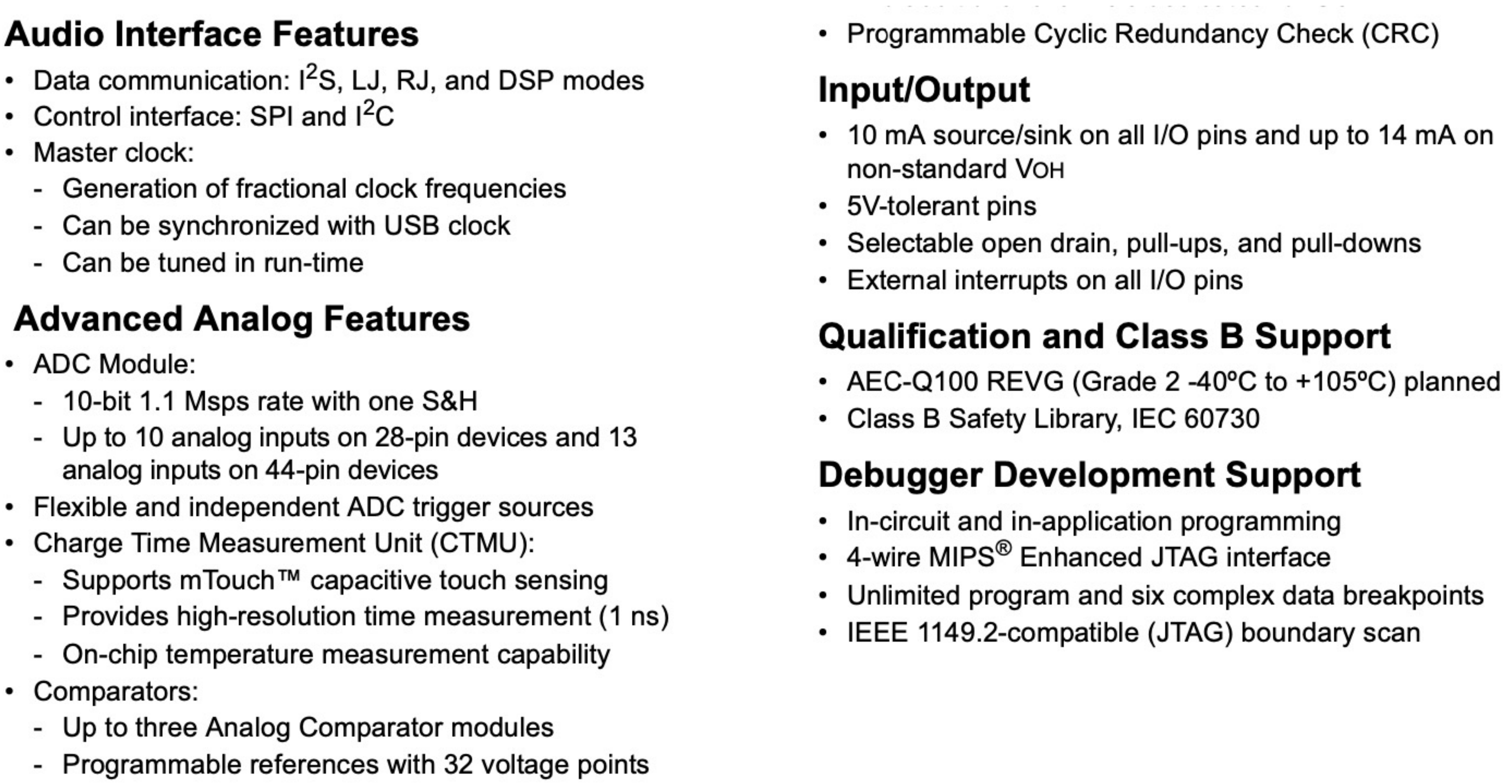
Communication Interfaces

- USB 2.0-compliant Full-speed OTG controller
- Two UART modules (12.5 Mbps):
  - Supports LIN 2.0 protocols and IrDA® support
- Two 4-wire SPI modules (25 Mbps)
- Two I^{2}C modules (up to 1 Mbaud) with SMBus support
- PPS to allow function remap
- Parallel Master Port (PMP)

Direct Memory Access (DMA)

- Four channels of hardware DMA with automatic data size detection
- Two additional channels dedicated for USB
- Programmable Cyclic Redundancy Check (CRC)

PIC32 MCU



Audio Interface Features

- Data communication: I^{2}S, LJ, RJ, and DSP modes
- Control interface: SPI and I^{2}C
- Master clock:
  - Generation of fractional clock frequencies
  - Can be synchronized with USB clock
  - Can be tuned in run-time

Advanced Analog Features

- ADC Module:
  - 10-bit 1.1 Msps rate with one S&H
  - Up to 10 analog inputs on 28-pin devices and 13 analog inputs on 44-pin devices
- Flexible and independent ADC trigger sources
- Charge Time Measurement Unit (CTMU):
  - Supports mTouch™ capacitive touch sensing
  - Provides high-resolution time measurement (1 ns)
  - On-chip temperature measurement capability
- Comparators:
  - Up to three Analog Comparator modules
  - Programmable references with 32 voltage points

- Programmable Cyclic Redundancy Check (CRC)

Input/Output

- 10 mA source/sink on all I/O pins and up to 14 mA on non-standard VOH
- 5V-tolerant pins
- Selectable open drain, pull-ups, and pull-downs
- External interrupts on all I/O pins

Qualification and Class B Support

- AEC-Q100 REVG (Grade 2 -40°C to +105°C) planned
- Class B Safety Library, IEC 60730

Debugger Development Support

- In-circuit and in-application programming
- 4-wire MIPS® Enhanced JTAG interface
- Unlimited program and six complex data breakpoints
- IEEE 1149.2-compatible (JTAG) boundary scan

PIC32 MCU

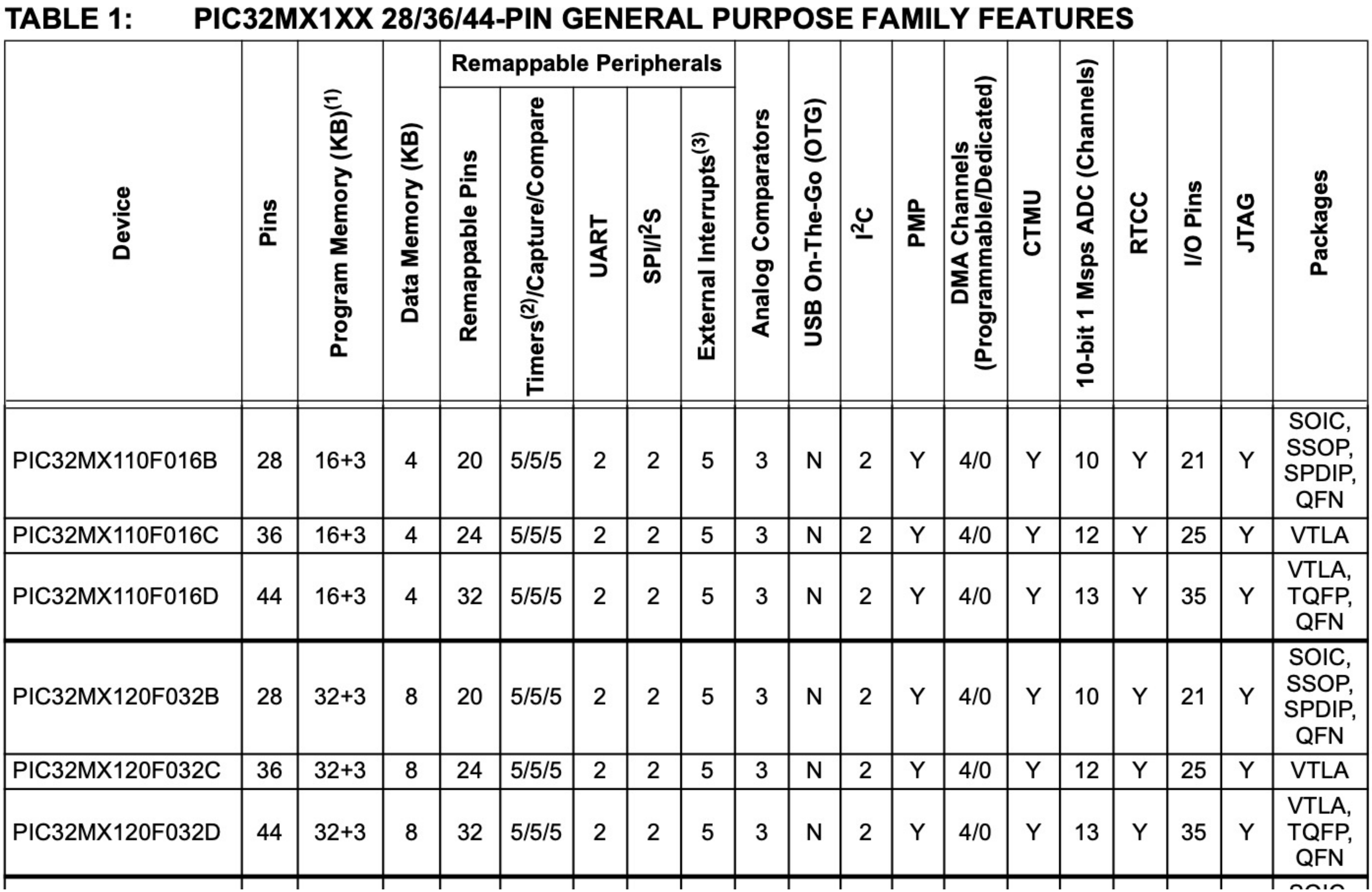
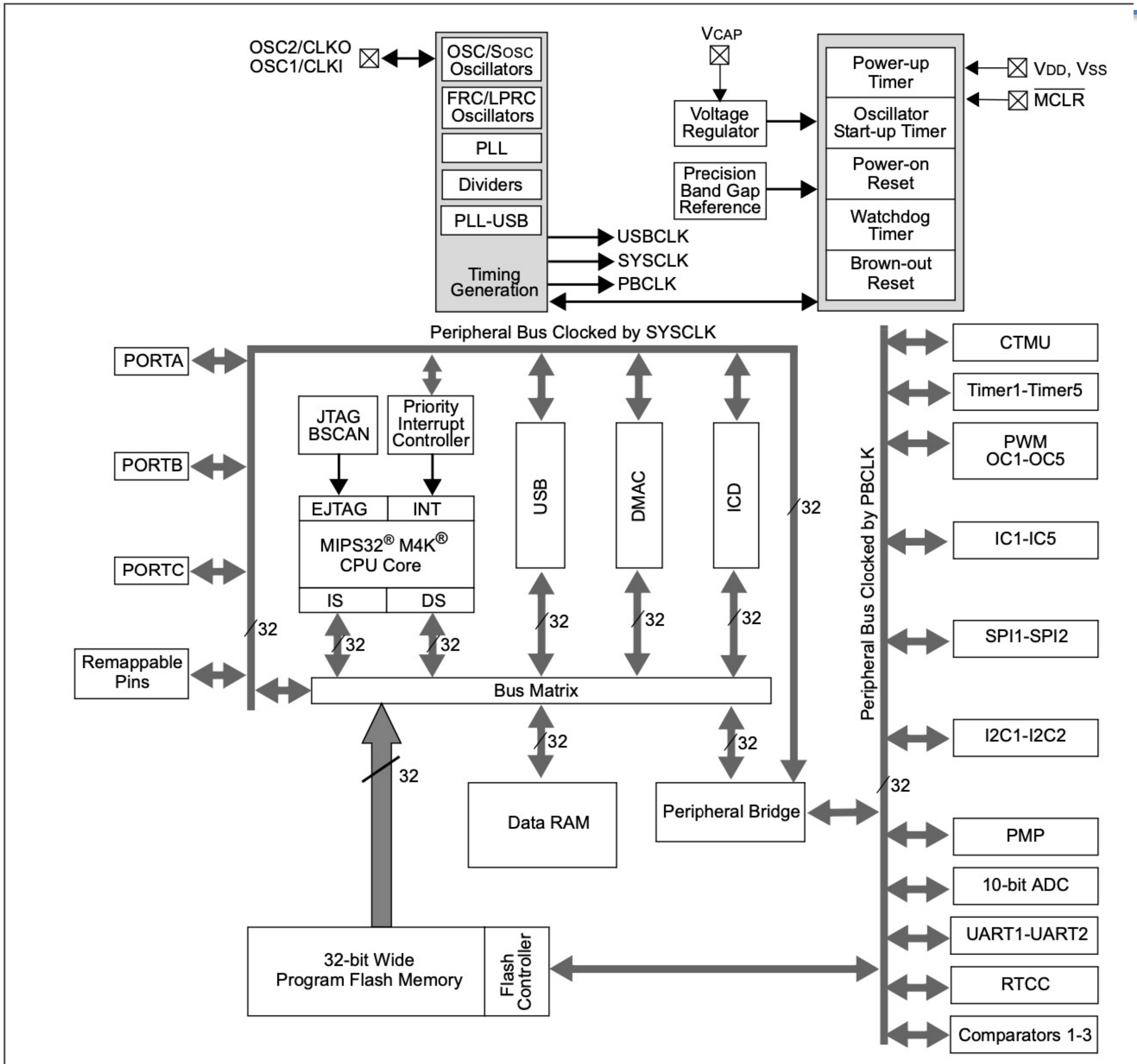


TABLE 1: PIC32MX1XX 28/36/44-PIN GENERAL PURPOSE FAMILY FEATURES

Device	Pins	Program Memory (KB)[1]	Data Memory (KB)	Remappable Peripherals					Analog Comparators	USB On-The-Go (OTG)	I^2C	PMP	DMA Channels (Programmable/Dedicated)	CTMU	10-bit 1 Msps ADC (Channels)	RTCC	I/O Pins	JTAG	Packages
				Remappable Pins	Timers[2]/Capture/Compare	UART	SPI/I^2S	External Interrupts[3]											
PIC32MX110F016B	28	16+3	4	20	5/5/5	2	2	5	3	N	2	Y	4/0	Y	10	Y	21	Y	SOIC, SSOP, SPDIP, QFN
PIC32MX110F016C	36	16+3	4	24	5/5/5	2	2	5	3	N	2	Y	4/0	Y	12	Y	25	Y	VTLA
PIC32MX110F016D	44	16+3	4	32	5/5/5	2	2	5	3	N	2	Y	4/0	Y	13	Y	35	Y	VTLA, TQFP, QFN
PIC32MX120F032B	28	32+3	8	20	5/5/5	2	2	5	3	N	2	Y	4/0	Y	10	Y	21	Y	SOIC, SSOP, SPDIP, QFN
PIC32MX120F032C	36	32+3	8	24	5/5/5	2	2	5	3	N	2	Y	4/0	Y	12	Y	25	Y	VTLA
PIC32MX120F032D	44	32+3	8	32	5/5/5	2	2	5	3	N	2	Y	4/0	Y	13	Y	35	Y	VTLA, TQFP, QFN

PIC32 MCU

FIGURE 1-1: BLOCK DIAGRAM

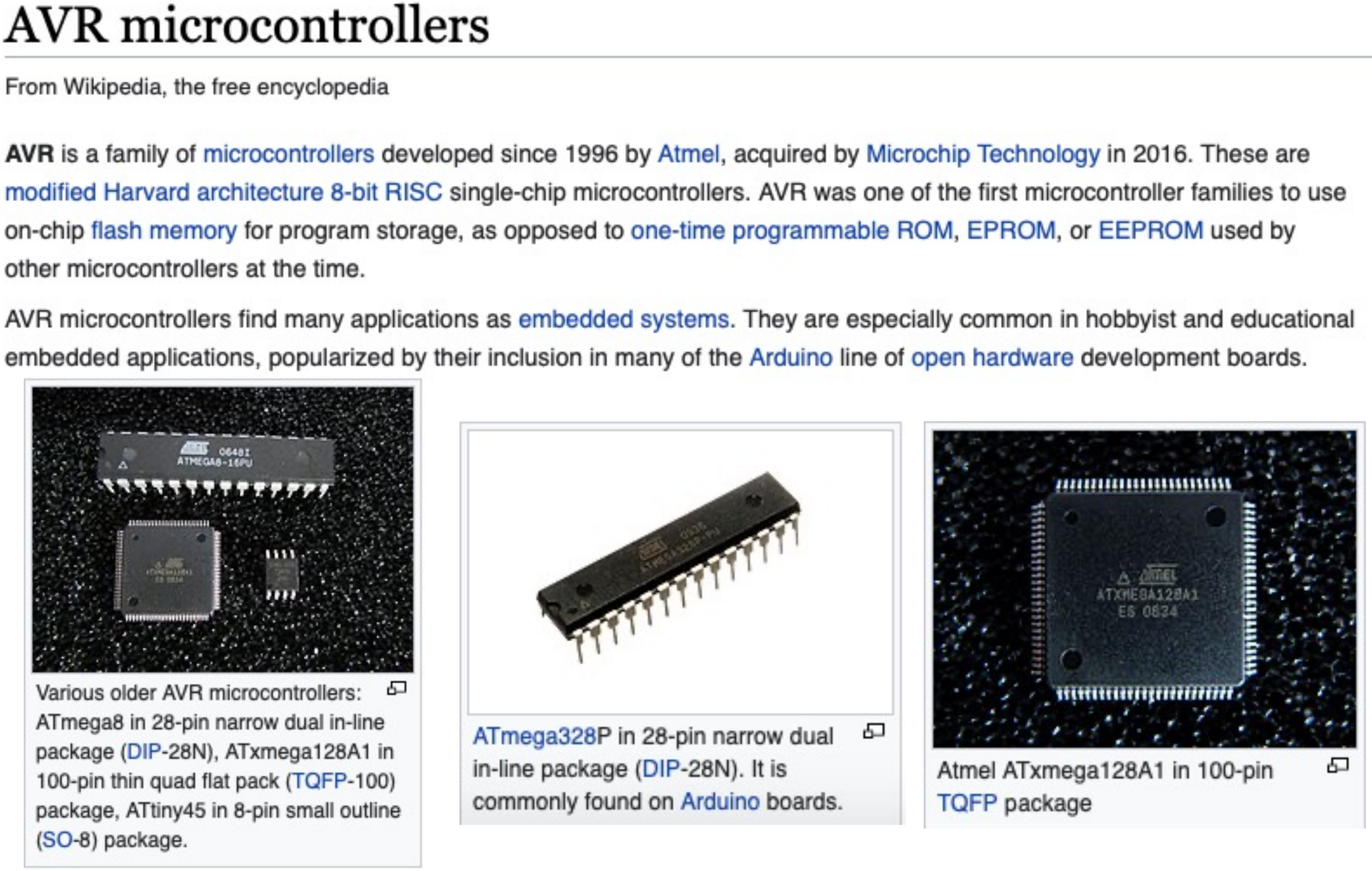


ISA

AVR

Atmel

AVR



AVR microcontrollers

From Wikipedia, the free encyclopedia

AVR is a family of microcontrollers developed since 1996 by Atmel, acquired by Microchip Technology in 2016. These are modified Harvard architecture 8-bit RISC single-chip microcontrollers. AVR was one of the first microcontroller families to use on-chip flash memory for program storage, as opposed to one-time programmable ROM, EPROM, or EEPROM used by other microcontrollers at the time.

AVR microcontrollers find many applications as embedded systems. They are especially common in hobbyist and educational embedded applications, popularized by their inclusion in many of the Arduino line of open hardware development boards.

Various older AVR microcontrollers: ATmega8 in 28-pin narrow dual in-line package (DIP-28N), ATxmega128A1 in 100-pin thin quad flat pack (TQFP-100) package, ATtiny45 in 8-pin small outline (SO-8) package.

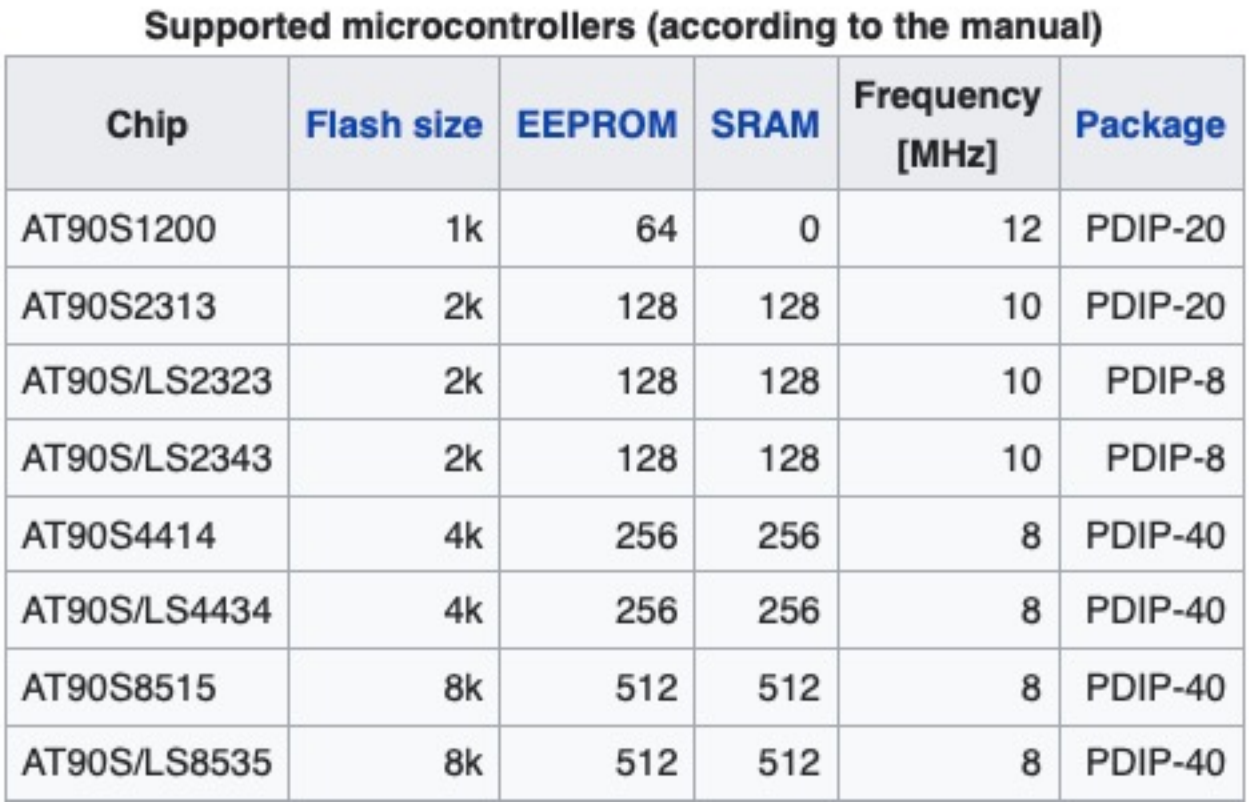
ATmega328P in 28-pin narrow dual in-line package (DIP-28N). It is commonly found on Arduino boards.

Atmel ATxmega128A1 in 100-pin TQFP package

AVR

Atmel AVR ATmega328 28-pin DIP on an Arduino Duemilanove board

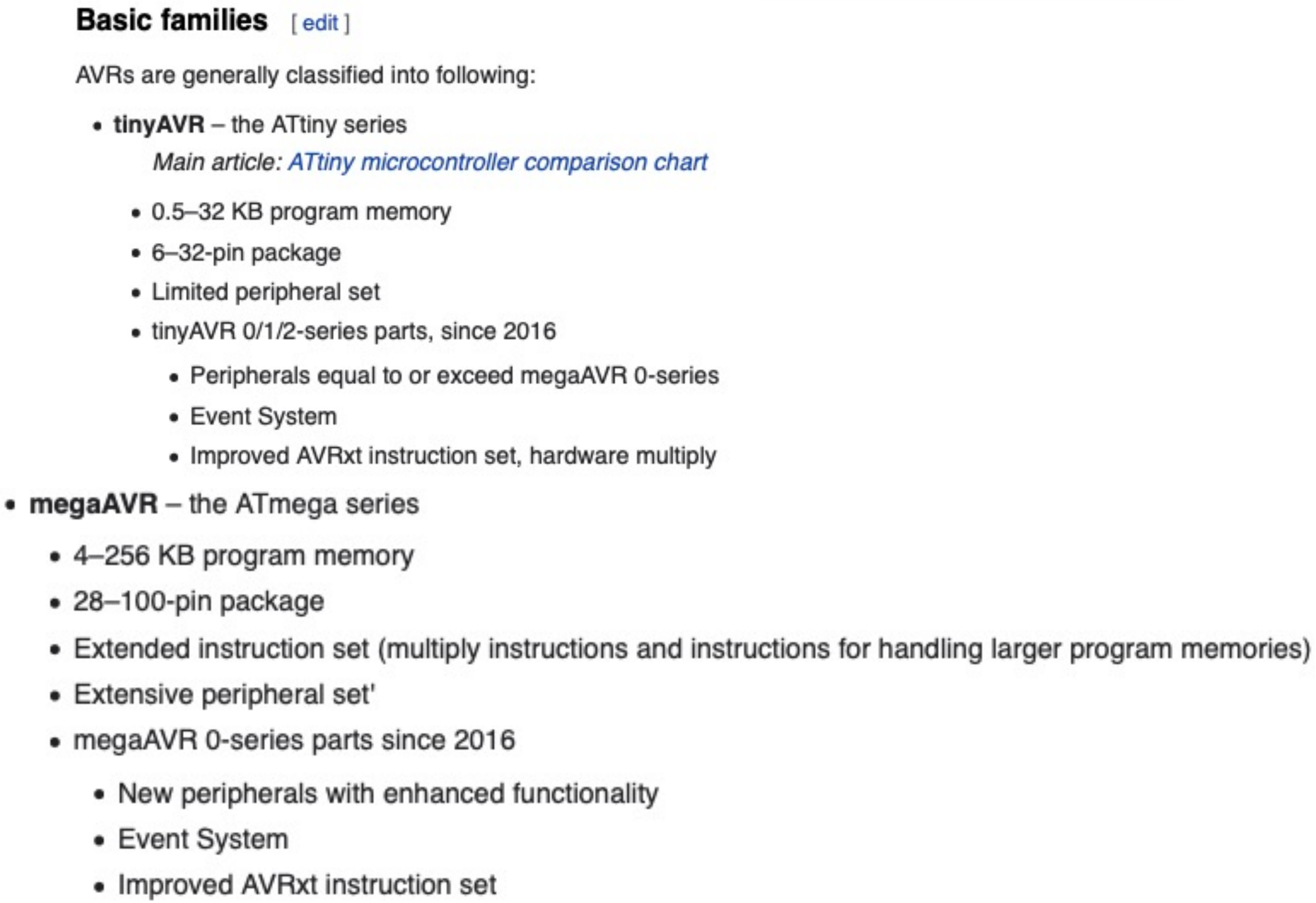
AVR



Supported microcontrollers (according to the manual)

Chip	Flash size	EEPROM	SRAM	Frequency [MHz]	Package
AT90S1200	1k	64	0	12	PDIP-20
AT90S2313	2k	128	128	10	PDIP-20
AT90S/LS2323	2k	128	128	10	PDIP-8
AT90S/LS2343	2k	128	128	10	PDIP-8
AT90S4414	4k	256	256	8	PDIP-40
AT90S/LS4434	4k	256	256	8	PDIP-40
AT90S8515	8k	512	512	8	PDIP-40
AT90S/LS8535	8k	512	512	8	PDIP-40

AVR



Basic families [edit]

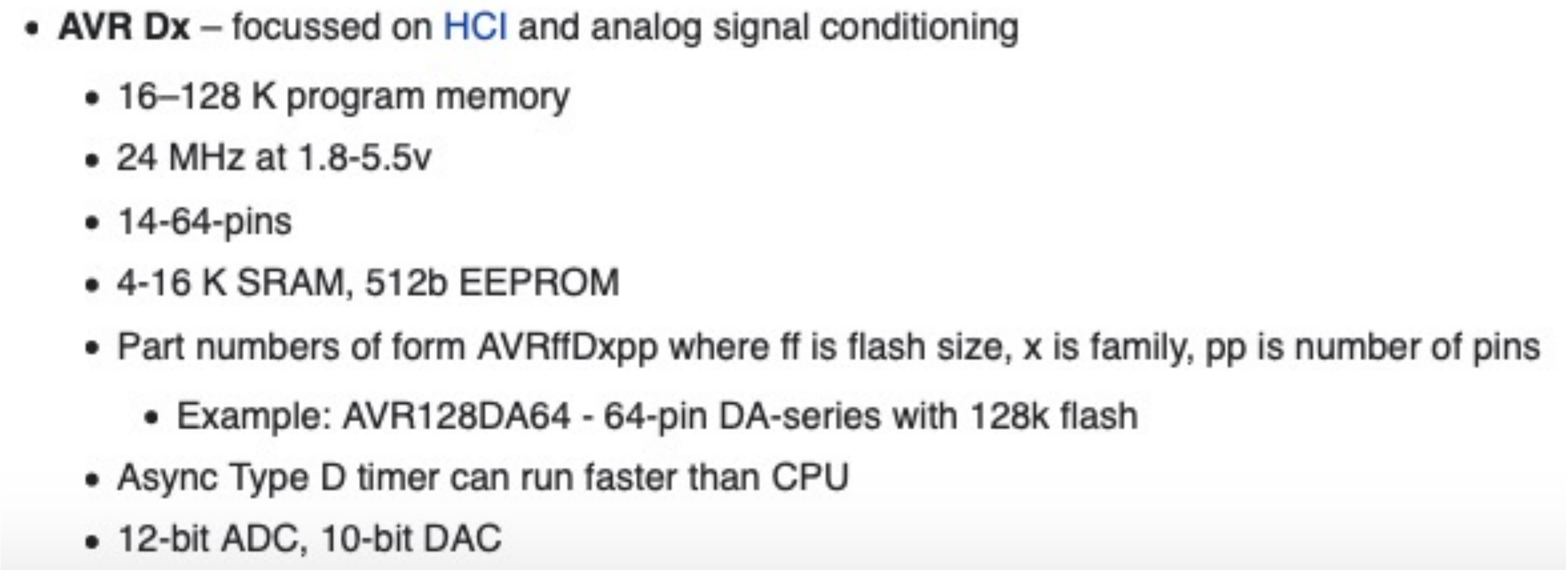
AVRs are generally classified into following:

- **tinyAVR** – the ATtiny series

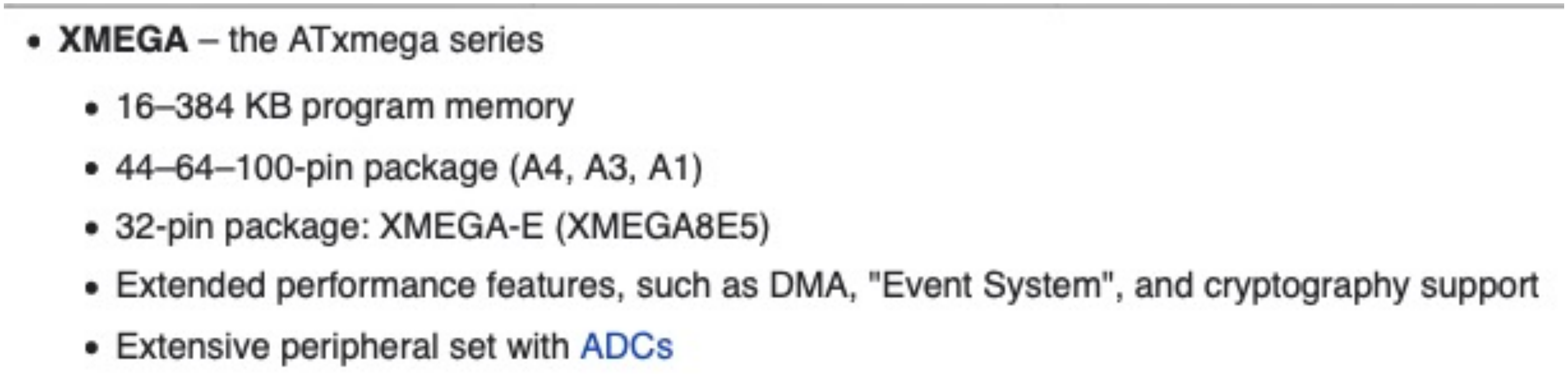
  Main article: ATtiny microcontroller comparison chart

  - 0.5–32 KB program memory
  - 6–32-pin package
  - Limited peripheral set
  - tinyAVR 0/1/2-series parts, since 2016
    - Peripherals equal to or exceed megaAVR 0-series
    - Event System
    - Improved AVRxt instruction set, hardware multiply
- **megaAVR** – the ATmega series
  - 4–256 KB program memory
  - 28–100-pin package
  - Extended instruction set (multiply instructions and instructions for handling larger program memories)
  - Extensive peripheral set'
  - megaAVR 0-series parts since 2016
    - New peripherals with enhanced functionality
    - Event System
    - Improved AVRxt instruction set

AVR

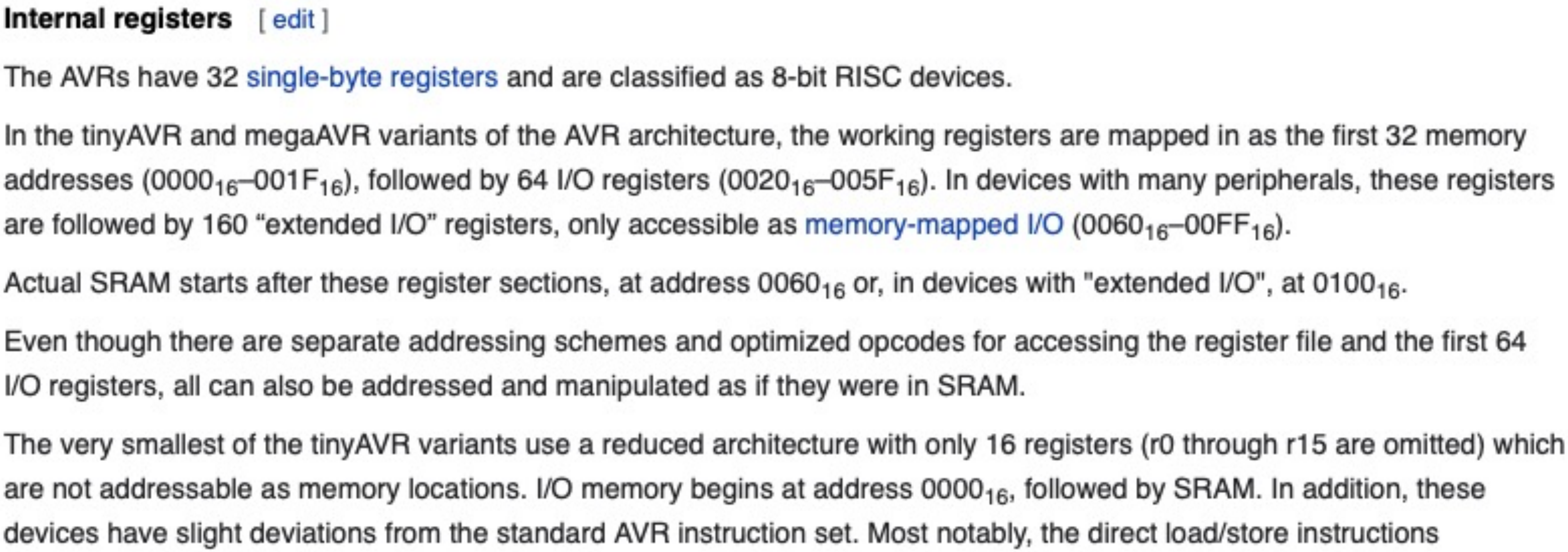


- **AVR Dx** – focussed on HCI and analog signal conditioning
  - 16–128 K program memory
  - 24 MHz at 1.8-5.5v
  - 14-64-pins
  - 4-16 K SRAM, 512b EEPROM
  - Part numbers of form AVRffDxpp where ff is flash size, x is family, pp is number of pins
    - Example: AVR128DA64 - 64-pin DA-series with 128k flash
  - Async Type D timer can run faster than CPU
  - 12-bit ADC, 10-bit DAC



- **XMEGA** – the ATxmega series
  - 16–384 KB program memory
  - 44–64–100-pin package (A4, A3, A1)
  - 32-pin package: XMEGA-E (XMEGA8E5)
  - Extended performance features, such as DMA, "Event System", and cryptography support
  - Extensive peripheral set with ADCs

AVR



Internal registers [edit]

The AVRs have 32 single-byte registers and are classified as 8-bit RISC devices.

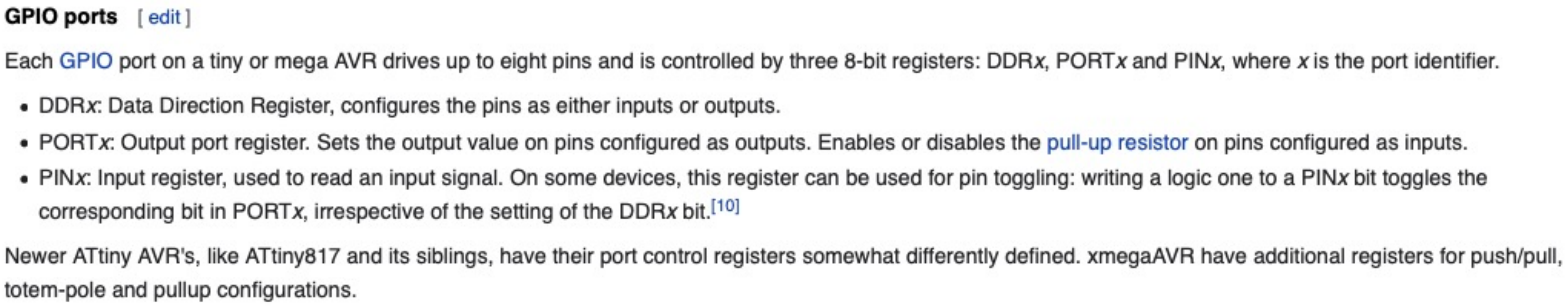
In the tinyAVR and megaAVR variants of the AVR architecture, the working registers are mapped in as the first 32 memory addresses (0000_{16}–$001F_{16}$), followed by 64 I/O registers (0020_{16}–$005F_{16}$). In devices with many peripherals, these registers are followed by 160 “extended I/O” registers, only accessible as memory-mapped I/O (0060_{16}–$00FF_{16}$).

Actual SRAM starts after these register sections, at address 0060_{16} or, in devices with "extended I/O", at 0100_{16}.

Even though there are separate addressing schemes and optimized opcodes for accessing the register file and the first 64 I/O registers, all can also be addressed and manipulated as if they were in SRAM.

The very smallest of the tinyAVR variants use a reduced architecture with only 16 registers (r0 through r15 are omitted) which are not addressable as memory locations. I/O memory begins at address 0000_{16}, followed by SRAM. In addition, these devices have slight deviations from the standard AVR instruction set. Most notably, the direct load/store instructions

AVR

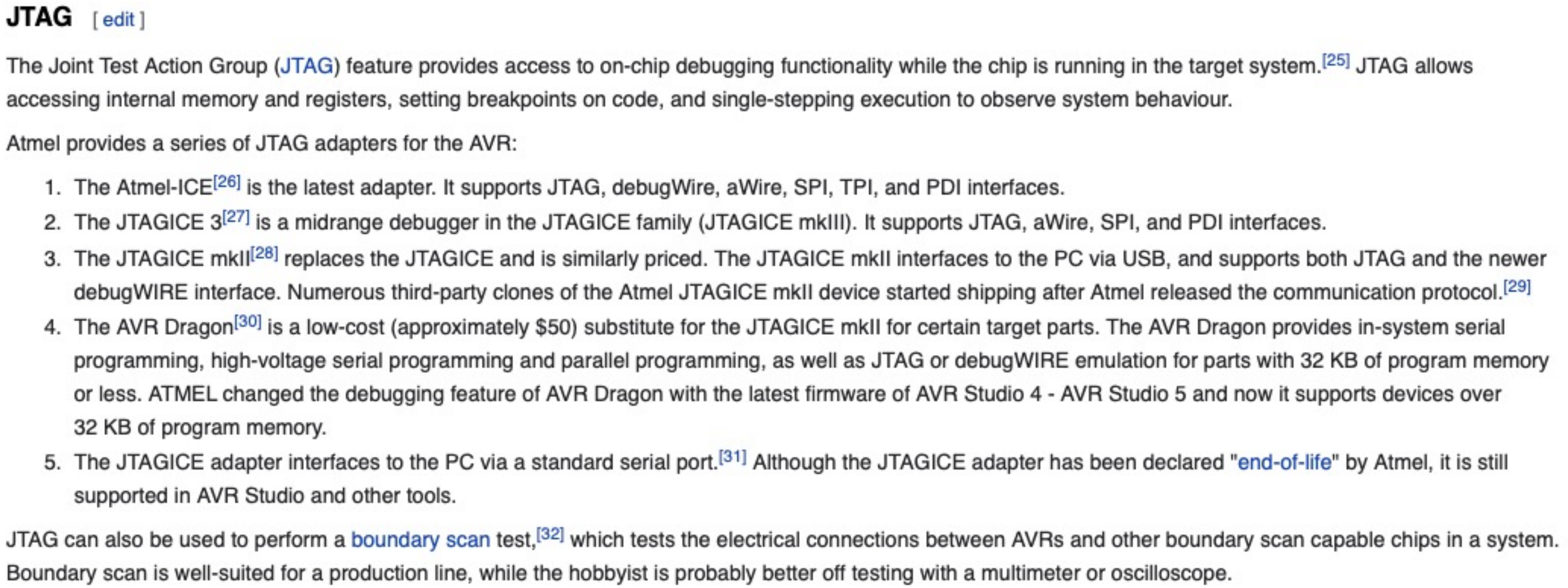


GPIO ports [edit]

Each GPIO port on a tiny or mega AVR drives up to eight pins and is controlled by three 8-bit registers: DDR*x*, PORT*x* and PIN*x*, where *x* is the port identifier.

- DDR*x*: Data Direction Register, configures the pins as either inputs or outputs.
- PORT*x*: Output port register. Sets the output value on pins configured as outputs. Enables or disables the pull-up resistor on pins configured as inputs.
- PIN*x*: Input register, used to read an input signal. On some devices, this register can be used for pin toggling: writing a logic one to a PIN*x* bit toggles the corresponding bit in PORT*x*, irrespective of the setting of the DDR*x* bit.[10]

Newer ATtiny AVR's, like ATtiny817 and its siblings, have their port control registers somewhat differently defined. xmegaAVR have additional registers for push/pull, totem-pole and pullup configurations.



JTAG [edit]

The Joint Test Action Group (JTAG) feature provides access to on-chip debugging functionality while the chip is running in the target system.[25] JTAG allows accessing internal memory and registers, setting breakpoints on code, and single-stepping execution to observe system behaviour.

Atmel provides a series of JTAG adapters for the AVR:

1. The Atmel-ICE[26] is the latest adapter. It supports JTAG, debugWire, aWire, SPI, TPI, and PDI interfaces.
2. The JTAGICE 3[27] is a midrange debugger in the JTAGICE family (JTAGICE mkIII). It supports JTAG, aWire, SPI, and PDI interfaces.
3. The JTAGICE mkII[28] replaces the JTAGICE and is similarly priced. The JTAGICE mkII interfaces to the PC via USB, and supports both JTAG and the newer debugWIRE interface. Numerous third-party clones of the Atmel JTAGICE mkII device started shipping after Atmel released the communication protocol.[29]
4. The AVR Dragon[30] is a low-cost (approximately $50) substitute for the JTAGICE mkII for certain target parts. The AVR Dragon provides in-system serial programming, high-voltage serial programming and parallel programming, as well as JTAG or debugWIRE emulation for parts with 32 KB of program memory or less. ATMEL changed the debugging feature of AVR Dragon with the latest firmware of AVR Studio 4 - AVR Studio 5 and now it supports devices over 32 KB of program memory.
5. The JTAGICE adapter interfaces to the PC via a standard serial port.[31] Although the JTAGICE adapter has been declared "end-of-life" by Atmel, it is still supported in AVR Studio and other tools.

JTAG can also be used to perform a boundary scan test,[32] which tests the electrical connections between AVRs and other boundary scan capable chips in a system. Boundary scan is well-suited for a production line, while the hobbyist is probably better off testing with a multimeter or oscilloscope.