80251 Microcontoller Jumps and Calls *4/12/2019*

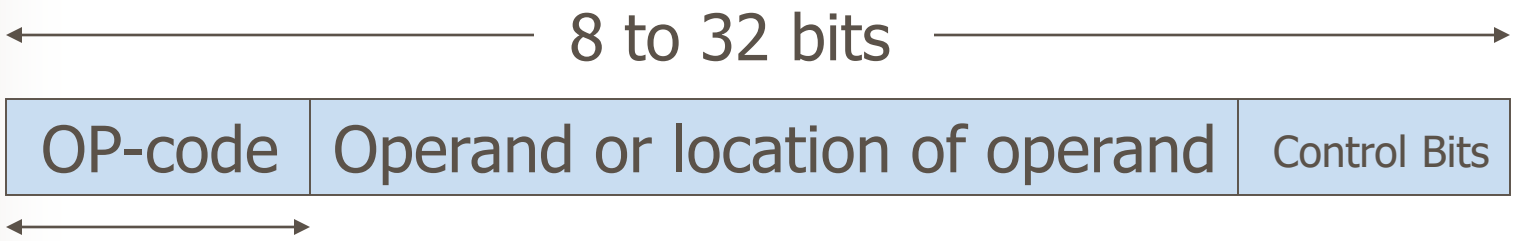
JT Wunderlich PhD

Recall Minimal Computer Architecture

- **PROGRAM COUNTER** (**PC**) addresses instructions to be fetched from memory
- **INSTRUCTION REGISTER** receives fetched instruction
- **CONTROL LOGIC** creates all routing signals after decoding the fetched instruction
- **ARITHMETIC LOGIC UNIT (ALU)** performs arithmetic and logical manipulation of data and addresses
- **REGISTERS** (i.e.,general purpose registers) store intermediate results of calculations
- **STATUS REGISTER** holds status flags and condition codes, and two control bits for picking register bank to be in use
- **"MEMORY"** (i.e. "main memory") stores data and instructions
- **STACK** stores addresses (or processor status) for returning from program-calls or interrupts
- **INPUT/OUTPUT** ("I/O") often addressed as memory (i.e., memory-mapped I/O)

Recall Minimal Computer Architecture

- Typical Instruction Format:

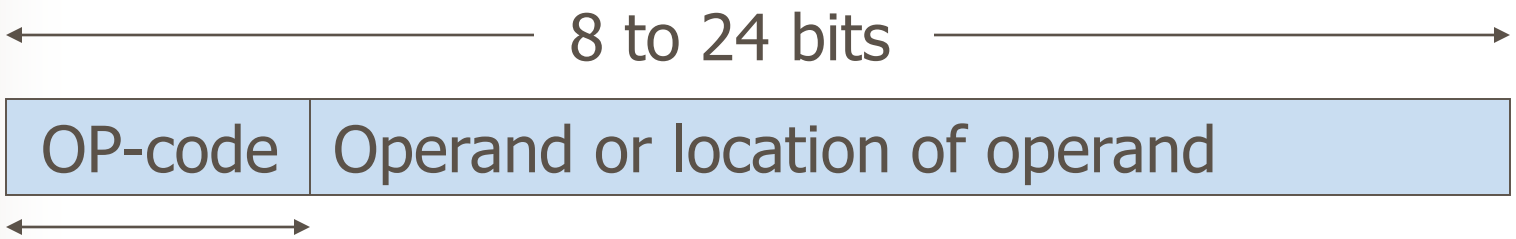


5 to 11 bits in OP-Code
Therefore 2^5=32 to 2^11=2048 different machine instructions in "**Instruction Set**"

- Some simple Microcontrollers (e.g., PIC's) have only 32 instructions
- Some large-scale machines (e.g., IBM S/390) have close to 2000 instructions

80251

- Typical Instruction Format:



Therefore 2^8=256 possible machine instructions in "**Instruction Set**"

80251 Memory Map

- Program Counter ("PC") is 24 bits
  - Therefore 2^24=16M of address space
    - 000000 to FFFFFF hex
  - Separated into 255 64K segments
    - **00**0000 to **FF**FFFF hex
      - Typically use segment 00 for all data manipulation
        - General purpose registers
        - Direct and indirectly addressable RAM
        - Bit addressable RAM
      - Typically use segment FF (i.e., #255) for all code and constants
  - 64K segments separated into 2K pages

Addressing Modes vs. "Range" classifications

- Modes
  1. Immediate
  2. Register
  3. Direct
  4. Indirect
  5. Indirect Offset

- Range
  1. "Short" (Relative)
  2. "Page" (Absolute)
  3. "Long" (Absolute)
     - Within Segment

Absolute just meaning not relative

Page = 2k (i.e., 2^11)

Segment = 64k (i.e., 2^16)

Addressing Modes vs. Ranges

- Modes
  1. Immediate *means operand data is encoded by you into instruction*
  2. Register *means operand data is in registers*
  3. Direct *means operand data is at location encoded by you into instruction*
  4. Indirect *means operand data is at location specified in a register*
  5. Indirect Offset *means operand data is at location specified in a register plus an "Offset" (encoded in instruction by you, or located in a register) that can be used, for example, to allow you to step through locations in an array*

Address RANGES

and Branch-target Address Formation

- "Short" (Relative)
  - (PC) ← (PC) + Instruction length + "Radd"
    - Radd = 1-byte relative address encoded into instruction (-128 to 127) in 2's compliment
- "Page" (Absolute *within 2^11 = 2K page*)
  - (PC) ← (PC) + Instruction length
  - (PC bits 11 to 0) ← "Sadd"
    - Sadd = Assembler uses 1-byte encoded into instruction to create an 11-bit Sadd (e.g, mapped to a label – assembler error if label on another page)
    - (PC bits 15 to 12) are fixed and indicate which page you are on
- "Long" (Absolute *within 2^16 = 64k segment*)
  - (PC) ← (PC) + Instruction length
  - (PC bits 15 to 0) ← "Ladd"
    - Ladd = 2-bytes encoded into instruction

Short (relative RANGE)

INSTRUC	When to Jump	Other Actions
JC	(C) = 1	
JNC	(C) not= 1	
JB	(bit address) = 1	
JNB	(bit address) not= 1	
JBC	(bit address) = 1	Make (bit address) = 0
CJNE	operand 1 not= operand 2	(C) = 1 if Op1 < Op2
DJNZ	operand 1 not= 0	Decrement first
JZ	(A) = 0	
JNZ	(A) not= 0	
SJMP	Always (i.e., “unconditionally”)	

Page (Absolute RANGE **within page**)

INSTRUC	When to Jump	Other Actions
AJMP	always	
ACALL	always	PUSH address of next instruction onto stack so you can return from this subroutine "Call" ((SP)) ← (PC) + instruction length
RET	always	POP address of next instruction off of stack so you can return from subroutine with this instruction last in it (PC) ← ((SP))

Long (to anywhere in space)

INSTRUC	When to Jump	Other Actions
LJMP	always	
LCALL	always	PUSH address of next instruction onto stack so you can return from this subroutine "Call" ((SP)) ← (PC) + instruction length
RET	always	POP address of next instruction off of stack so you can return from subroutine with this instruction last in it (PC) ← ((SP))

Special Case to access **external memory** space

INSTRUC	When to Jump	Other Actions
JMP	always	(PC) ← (A) + (DPTR) DPTR is a 16 bit pointer Ground “EA” pin on chip to force all code fetches to be from external memory (via port pins)
RET	always	POP address of next instruction off of stack so you can return from subroutine with this instruction in it (PC) ← ((SP))

Compare RANGE's

- "Relative-Range"
  1. Only one byte for branch target formation
  2. Limited jump range (-128 to 127)
  3. If relocating code, watch page-boundaries
- "Short-Absolute" (Absolute *within 2^11 = 2K page*)
  1. Only one byte needed to begin branch target formation
  2. Better jump range (2K of options)
  3. If relocating code, watch page-boundaries
- "Long-Absolute"
  1. Two bytes needed for branch target formation
  2. Best jump range (64K of options)

Session 3547

Focusing on the Blurry Distinction between Microprocessors and Microcontrollers

J. T. Wunderlich PhD
Elizabethtown College (present) and Purdue University (1999)

Abstract

This paper compares microprocessors and microcontrollers in the context of teaching a sophomore level course where students have completed previous studies in digital circuits and programming. Discussing the similarities between these devices helps reinforce the understanding of the basic function of either device. Topics such as the "fetch-decode-execute" of an instruction cycle, or the memory-mapping of I/O provide good examples of similarities. Discussing the differences helps identify which device is most suitable for a given application. Topics such as mathematical computation capabilities or the ability to contain all needed functionality on a single chip provide good examples of differences. It is also important to study these devices in the context of historical trends since today's microcontrollers have evolved from past microprocessors. The microcontroller of the future could look more like today's microprocessors -- with a wider data bus, enhanced mathematical functionality, and numerous speed-up schemes. However, many of the unique features of microcontrollers are unlikely to be found in future microprocessors -- the separate memory for instructions and data is one example; the on-chip I/O control features such as analog-to-digital conversion and pulse-width-modulated outputs are other examples. The understanding of microprocessors and microcontrollers can also be enhanced by considering the differences between how programmers and engineers may view these devices. For example, a device could be selected for the programming power of the instruction-set, or for the simplicity of the instruction-set and minimization of additional circuitry.

Proceedings of the 1999 American Society for Engineering Education Annual Conference & Exposition

Figure 2. Example MC68000 microprocessor program using 16-bit arithmetic to do a 16-bit task; Decrement the 16-bits in general-purpose data register D0 until it reaches the 16-bit number in general-purpose data register D2.

LINE				# OF BYTES	# OF CYCLES
01	check:	CMP.W D0, D2	; compare D0 and D2, set appropriate condition flag	2	4
02		DBE D0, check	; decrement, and jump to " check " until D0 and D2 equal	4	10 to 12
03	done:	NOP	; program finished	2	4
				=====	
			TOTAL =	8	

Figure 3. Example 8051 microcontroller program using 8-bit arithmetic to do a 16-bit task; Decrement the 8-bit general-purpose registers R1 and R0 as one concatenated 16-bit number until it reaches the 16-bit number made by concatenating the contents of the 8-bit general-purpose registers R3 and R2.

LINE				# OF BYTES	# OF CYCLES
00	check:	MOV A, R0	;put low-order byte in accumulator	1	1
01		CJNE A, 02h, dcrmnt	;conditional jump to "dcrmnt" if not equal to R2 contents	3	2
02		MOV A, R1	;put high-order byte in accumulator	1	1
03		CJNE A, 03h, dcrmnt	;conditional jump to " dcrmnt " if not equal to R3 contents	3	2
04		SJMP done	;countdown finished, jump to "done"	2	2
05	dcrmnt:	MOV A, R0	;put low-order byte in accumulator	1	1
06		CLR C	;must clear carry flag since used in subtraction	1	1
07		SUBB A, #01h	;decrement (and possibly set borrow)	2	1
08		MOV R0 ,A	;temporarily store new high-order byte in R0	1	1
09		MOV A, R1	;put high-order byte in accumulator	1	1
10		SUBB A, #00h	;subtract borrow (i.e., carry bit is set if borrow at line #07)	2	1
11		MOV R1 ,A	;temporarily store new high-order byte in R1	1	1
12		SJMP check	;jump to "check "	2	2
13	done:	NOP	;program finished	1	1
				=====	
			TOTAL =	22	

Figure 4. Example MC68000 microprocessor program using 16-bit arithmetic to do a 16-bit task; Decrement the 16-bits at RAM location 2000h until it reaches the 16-bit number in general-purpose data register D2; then store count back into memory.

LINE				# OF BYTES	# OF CYCLES
00		MOVE.W $2000, D0	; copy original count into register D0 from RAM (off-chip)	4	12
01	check:	CMP.W D0, D2	; compare D0 and D2, set appropriate condition flag	2	4
02		DBE D0, check	; decrement, and jump to " check " until D0 and D2 equal	4	10 to 12
03		MOVE.W D0, $2000	; write count to RAM (off-chip) from D0	4	12
04	done:	NOP	; program finished	2	4
				=====	
			TOTAL =	16	

Figure 5. Example 8051 microcontroller program using 8-bit arithmetic to do a 16-bit task; Decrement the 8-bit contents of internal RAM addresses 21h and 20h as one concatenated 16-bit number until it reaches the 16-bit number made by concatenating the contents of the 8-bit general-purpose registers R3 and R2 [2].

LINE				# OF BYTES	# OF CYCLES
00	check:	MOV A, 20h	;get low-order byte from on-chip RAM	2	1
01		CJNE A, 02h, dcrmnt	;conditional jump to "dcrmnt" if not equal to R2 contents	3	2
02		MOV A, 21h	;get high-order byte from on-chip RAM	2	1
03		CJNE A, 03h, dcrmnt	;conditional jump to " dcrmnt " if not equal to R3 contents	3	2
04		SJMP done	;countdown finished, jump to "done"	2	2
05	dcrmnt:	MOV A, 20h	;get low-order byte from on-chip RAM for decrementing	2	1
06		CLR C	;must clear carry flag since it is used as a borrow	1	1
07		SUBB A, #01h	;decrement (and possibly set borrow)	2	1
08		MOV 20h ,A	;store new high-order byte in on-chip RAM	2	1
09		MOV A, 21h	;get high-order byte from on-chip RAM for decrementing	2	1
10		SUBB A, #00h	;subtract borrow (i.e., carry bit is set if borrow at line #07)	2	1
11		MOV 21h ,A	;store new low-order byte in on-chip RAM	2	1
12		SJMP check	;jump to "check "	2	2
13	done:	NOP	;program finished	1	1
				=====	
			TOTAL =	28	