CpE 213
Digital Systems Design

Lecture 8

Thursday 9/18/2003

Before the next lecture

- Review Chapter 2.
- Read Chapter 3.

Program Status Word (PSW)

- SFR at address D0H

MSB LSB

CY	AC	F0	RS1	RS0	OV	—	P

BIT	SYMBOL	FUNCTION
PSW.7	CY	Carry flag.
PSW.6	AC	Auxilliary Carry flag. (For BCD operations.)
PSW.5	F0	Flag 0. (Available to the user for general purposes.)
PSW.4	RS1	Register bank select control bit 1. Set/cleared by software to determine working register bank. (See Note.)
PSW.3	RS0	Register bank select control bit 0. Set/cleared by software todetermine working register bank. (See Note.)
PSW.2	OV	Overflow flag.
PSW.1	—	User-definable flag.
PSW.0	P	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of "one" bits in the Accumulator, i.e., even parity.

NOTE: The contents of (RS1, RS0) enable the working register banks as follows:
(0,0)— Bank 0 (00H–07H)
(0,1)— Bank 1 (08H–0fH)
(1,0)— Bank 2 (10H–17H)
(1,1)— Bank 3 (18H–17H)

SU00531A

More SFRs

- B Register
  - used with ACC for multiplication and division
  - can also be a general-purpose scratch-pad
  - is bit-addressable
- Stack Pointer (SP)
  - contains address of data item on top of stack
  - stack operations
    - push (increments SP)
    - pop (decrements SP)
  - stack is in internal RAM
  - limited to addresses accessible through indirect addressing

Even more SFRs

- Data Pointer (DPTR)
  - used to access external code or data memory
  - 16-bit register at 82h(low byte) and (83H)
  - to write 65H into external RAM location 1000H

    MOV A, #55H

    MOV DPTR, #1000H

    MOVX @DPTR, A

Port Registers

- Port 0 at address 80H
- Port 1 at address 90H
- Port 2 at address A0H
- Port 3 at address B0H
- Ports 0,2,3 are not always available for I/O
- P1.2 through P1.7 are always available for I/O
- All ports are bit-addressable

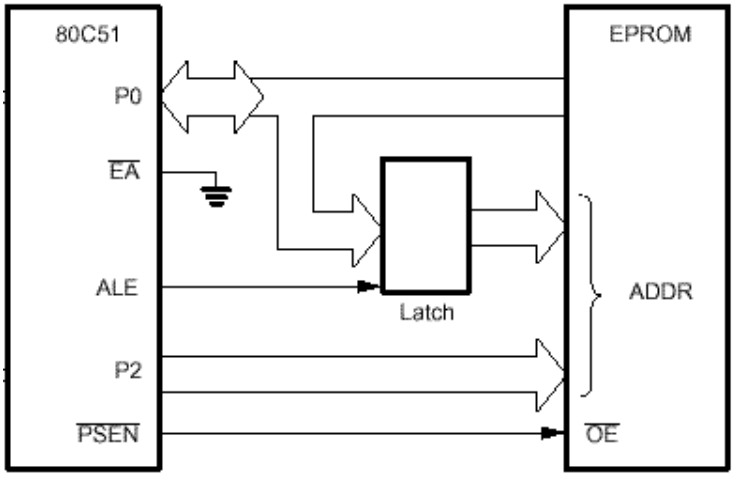
Other SFRs (read from textbook)

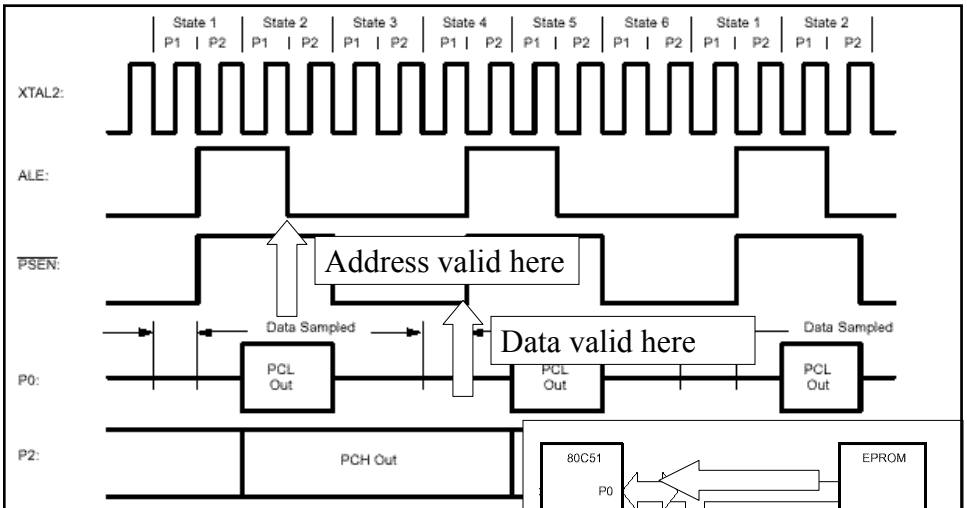
- Timer-related:
  - Two timer registers
  - Timer mode register
  - Timer control register
- Serial port registers
- Interrupt registers
- Power Control register

External Memory

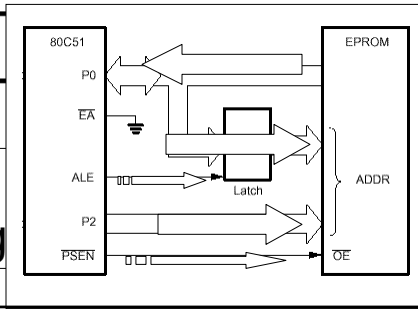
- 64K external code memory space
- 64K external data memory space
- When external memory is used:
  - Port 0 becomes muxed address and data bus
  - ALE latches low byte of address at beginning of each external memory cycle
  - Port 2 is usually used for high byte of address bus
  - Ports 0 and 2 become unavailable as general purpose I/O ports
  - Muxing saves us 8 bits that can be used to offer other functions

External external code memory

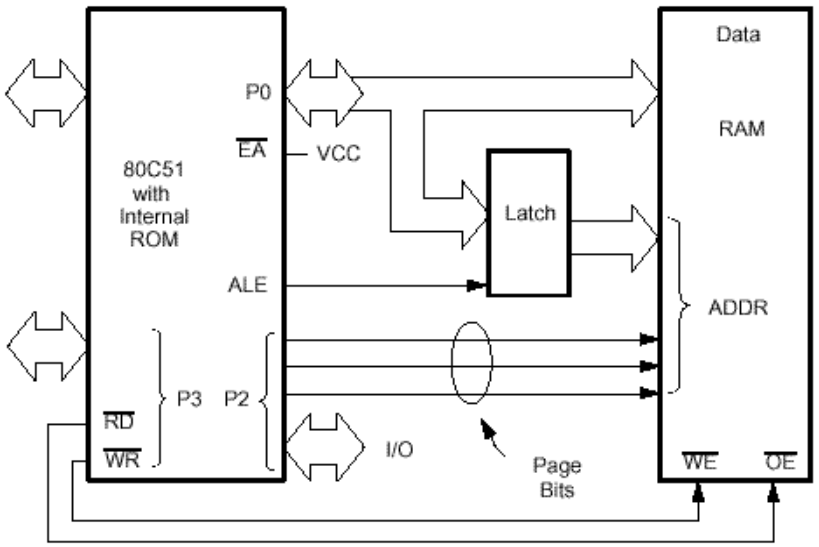




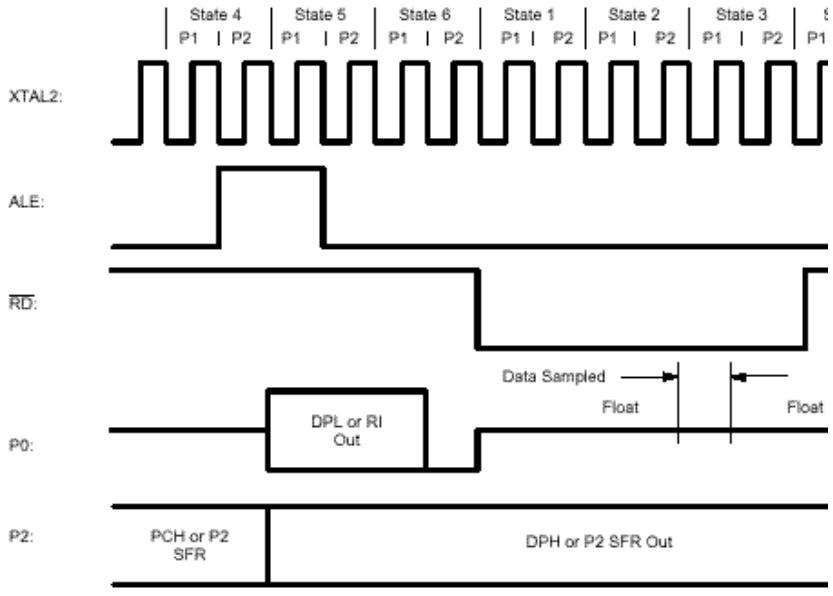
Instruction Fetch Timing



External Data memory



External Data (read or write)



8051 Instruction Set
Chapter 3

Machine Instructions (MI)

- These are the 8051's primitive operations
- They are the computer's **Instruction Set**
- MI = Operation + Operand(s)
- Unary instruction = Op(x)
  - CPL A ; complement the Accumulator register
- Binary instruction: x Op y (or Op x,y)
  - ADD A,#10 ; add ten to the accumulator

Machine Instructions (MI)

- A typical **Expression**:
  x= y+z;
- 'x' is the name of a **Destination**
- 'y' and 'z' are names of two data **Sources**
- Needs an **opcode** (add) plus three **addresses:**

Op	Dest	Src1	Src2

Machine Instructions (MI)

- Four fields are too many for 8 bit microcontroller: 4+16+16+16 = 52 bits!
- Other alternatives:
  - two address: x=x+y

add	x	y

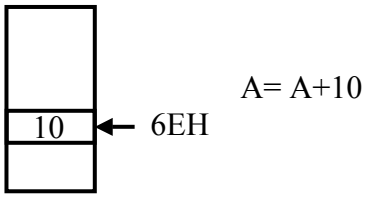
  - one address: A=A+x

add	x

- Most 8051 instructions are single address
- The Accumulator (A) is an *implied* address.

Address Modes

- Implied (encoded in the opcode) (clr c)
- Immediate (operand follows the opcode)
  - Add A,#6EH ; A= A+110.
- Direct address (pointer to operand in memory)
  - Add A,6EH ;
  - Operand in on-chip data space

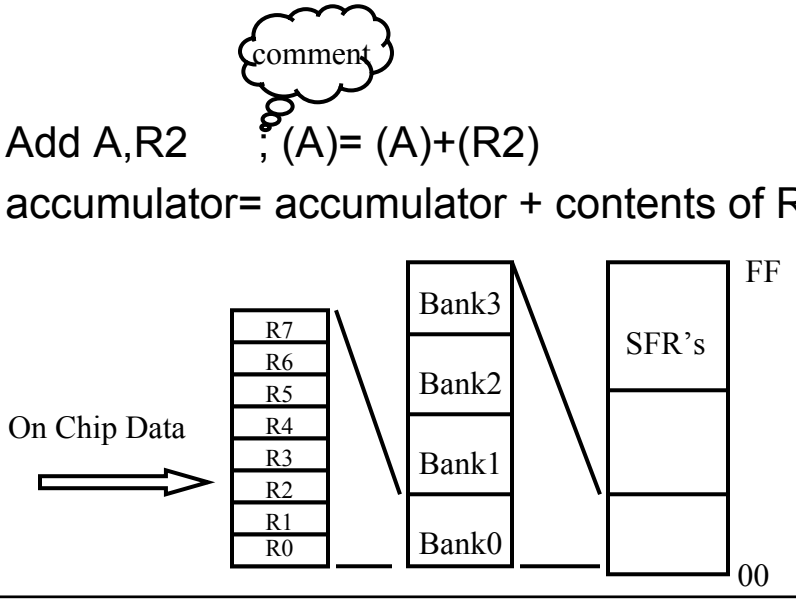


Address Modes

- Register direct (MOV A,R5)
- Register Indirect (MOV A,@R0 MOV A,@DPTR)
- Indexed (MOVC A,@A+DPTR)
- Relative (SJMP)
- Absolute
  - 11 bit (ACALL)
  - 16 bit (LJMP)

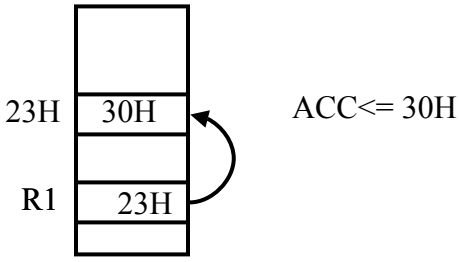
Register Addressing

- Add A,R2 ; (A)= (A)+(R2)
- accumulator= accumulator + contents of R2



Register Indirect

- MOV A,@R1
- Load ACC with the value in *data* space pointed to by the contents of R1
- Similar to the C assignment: achar = *cptr
- The *effective address* is equal to (R1) or 23H.



Indexed Addressing

CLR A ; ACC <= 0

MOV DPTR,#130H ; DPH<=01, DPL<=30

MOVC A,@A+DPTR

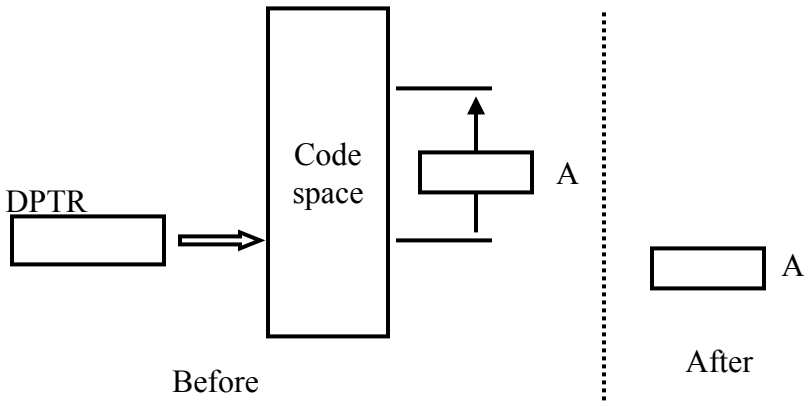
- load ACC with contents of code space pointed to by the sum of ACC plus DPTR.

MOVC A,@A+PC

- load ACC with contents of code space pointed to by the sum of ACC plus program counter.

Indexed Addressing (example)

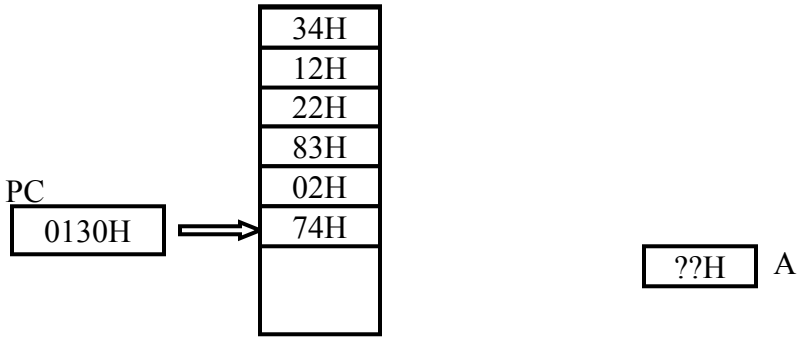
MOVC A,@A+DPTR ; assume acc contains 12H



Indexed Addressing (example)

74 02 **MOV A,#2** ; load ACC with 2

83 MOVC A,@A+PC ; ACC<= 0??H



Relative Addressing

- Only for branch instructions
- *Relative* to the program counter
- Example: SJMP $+3
  - **$ $+3**
  - **80 01 B0 B1 B2 …**

```
…
N    80 01 sjmp $+3
N+2  00    nop
N+3  00    nop
…
```

- Displacement (01) added to PC
- '$' is just a label for 'here'
  - Substitute 'here' for $ and 'there' for $+3
  - Labels are preferred. Avoid use of $.

Putting relative addresses to work

- Generate a square wave on P1.2
- Assume 12 Mhz clock
  - What is square wave period?
  - What is its duty cycle?

```
Loop:   setb p1.2
        mov r1,#3
        djnz r1,$
        clr p1.2
        mov r1,#4
        djnz r1,$
        sjmp loop
```

Relative Addressing

- Branch to self
  - Loop: SJMP loop ; infinite loop!
  - Encoded as: 80 FE
  - Displacement is –2
  - A better (more common) example:

Instruction Encoding

- Multiples of 8 bit bytes
- Most are one byte, some two, a few three.
- Encoding of the OPCODE and ADDRESS
- Several examples follow

Question:

How many instructions are there?

a) <256 b) 256 c) >256

Simple example: CLR C

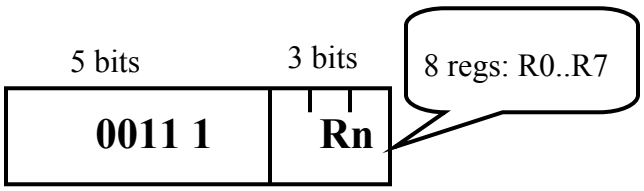
- Clear the carry bit
- 8 bit opcode
- Address is implied
- CLR C is called *assembly code* or *symbolic code*
- *Machine code* is 0C3H or 1100_0011B
- Assembly program listing format:

```
0130 C3 CLR C  ; clear carry
```

1100 0011

Better example: ADDC A,Rn

- Add ACC plus carry bit plus contents of Rn
- Put result into ACC
- Specify one operand and imply the other two



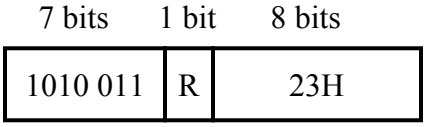

```
0130 39  ADDC a,R1
```

A similar example:
MOV @R0,23H

- What does it do?

A similar example:
MOV @R0,23H

- What does it do?
- Two byte encoding:
  (what is implied by this code?)




```
0130 A6 23  MOV @R0,23H
```

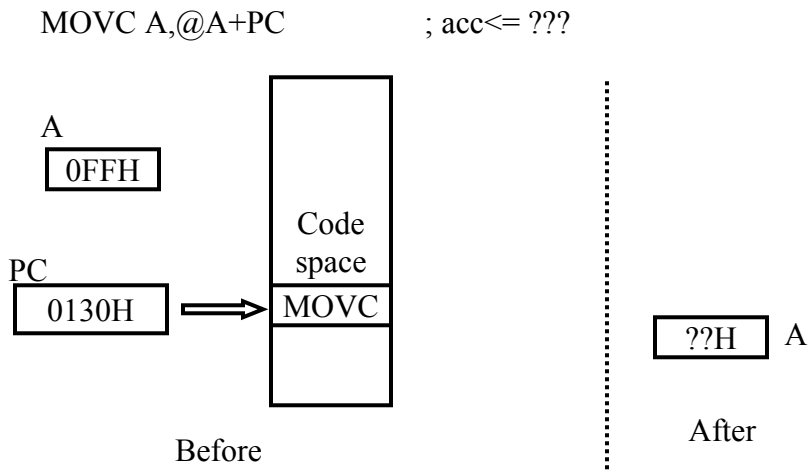
More complex example
AJMP next

- Absolute jump
- Can jump anywhere in a 2k byte page
- Encoding: 

- What about other five bits?
  (11 + 5 = 16 bits)

```
C800 C1 C2  AJMP next   ; PC<- ????
 A) pc<- 06C2h  B) something else
```

MOVC example
(what does this do?)



Instruction Timing

- Most are one cycle (12 clock periods)
- A few are two cycle (24 clocks)

Question:
Are there any three cycle instructions?

Instruction Timing

- Most are one cycle (12 clock periods)
- A few are two cycle (24 clocks)
- No three cycle instructions!
- MUL and DIV are four cycles (48 clocks)

Other instructions

- Data movement
- Exchange byte
- Stack operations
- Unconditional and conditional branches
- Subroutine call
- Arithmetic and Logical operations
- Bit manipulation (unique to 8051!)

Data Movement

- MOV

	A	#data	direct	@Ri	Rn	Src
A	✗	✓	✓	✓	✓	
direct	✓	✓	✓	✓	✓	
@Ri	✓	✓	✓	✗	✗	
Rn	✓	✓	✓	✗	✗	
Dst						

  - MOVC A,@A+DPTR (or PC)
  - MOVX A,@DPTR (or @DPTR,A)

Exchange Byte

- XCH A,direct
  - exchange A with a byte in data space
- XCH A,@Ri
  - exchange A indirect with byte in data
- XCH A,Rn
  - exchange A with one of 8 registers

Stack Operations

- PUSH direct
  - PUSH 25
    (SP) ← (SP)+1
    ((SP)) ← (25)
  - increment SP and store contents of location 25 in contents of location pointed to by SP.

```
Similar to C:
  unsigned char *sp, x;
  *(++sp) = x;
```

- POP direct
  - POP 25
    (25) ← ((SP))
    (SP) ← (SP)-1

```
Similar to C:
  unsigned char *sp, x;
  x= *(sp--);
```

Unconditional Branch

- LJMP addr16
  - LJMP 0132h (PC) ← 0132h
- SJMP rel
  - SJMP 0FCh (PC) ← (PC)-4
  - Operand is treated as a **signed** number and added to the PC
- AJMP addr11 see pgmrs guide
- JMP @A+DPTR (PC) ← (A)+(DPTR)

Conditional Branch

- Opcodes
  - JZ, JNZ, JC, JNC, JB, JNB, CJNE, DJNZ, JBC
- Relative Addressing
  - rel = *relative address*
  - 8 bit signed displacement from program counter
  - example: JZ 25
    (PC) ← (PC)+2
    if A=0 then (PC) ← (PC)+25

CJNE

- Compare and jump not equal
  - Compare A and direct byte or
  - #data and A, Rn or @Ri
- Example
  - CJNE R0,#25,-5
    (PC) ← (PC)+3
    if (R0)<>25 then (PC) ← (PC)-5
    if (R0)<25 then (C) ← 1 else (C) ← 0
    encoded as: `B8 19 FB`

DJNZ

- Decrement byte and Jump if Not Zero
  - decrement register or… (2 byte instruction)
  - direct byte (3 byte instruction)
- Example
  - DJNZ cnt,loop ; cnt=cnt-1, if cnt!=0 goto loop
  - assume loop at 100h, cnt at 20h, djnz at 110h
  - 100h - (110h+3) = 100h-113h = -13h = 0EDh
  - instruction coded as: D5 20 ED

JBC

- **J**ump if **B**it set and **C**lear bit
- This is an *atomic* test and modify instruction
- Example
  - Let (A)=56h=01010110, loop=100h, jbc at 110h
    JBC A.3,loop ; does not jump
    JBC A.2,loop ; jumps to loop with (A)=52h
    coded as: 10 E2 ED
  - E2 is the address of ACC bit 2. ACC is loc E0h

Subroutine Call

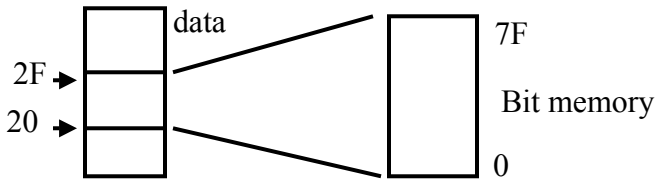
- LCALL addr16 - similar to LJMP
  - (PC) ← (PC)+3
  - (SP) ← (SP)+1
  - ((SP)) ← PCL (PC(7:0) or PC low)
  - (SP) ← (SP)+1
  - ((SP)) ← PCH (PC(15:8) or PC high)
  - (PC) ← addr16
  - **`LJMP 0ABCDh`** is encoded as: **`12 AB CD`**
- ACALL - similar to AJMP

Arithmetic and Logical Operations

- ANL, ORL, XRL, CPL, CLR
  - and, or, xor, 1s' complement, clear
- ADD, ADDC
- SUBB (no SUB)
- INC, DEC, INC DPTR (no DEC dptr)
  - typically A, #, direct, @Ri, and Rn modes
- XCHD, SWAP, DA (bcd instructions)
  - xchd A,@Ri, Swap A, and DA A only

Bit Manipulation

- Powerful feature of 8051 family
- CLR, SETB, CPL, ANL, ORL, MOV
- C acts as a bit ‘accumulator’
- Uses C, 128 ram bits, and many SFR’s
- Ram bits at locations 20h through 2Fh



Assembly code

- Looking up opcodes is tedious and error prone
- Solution: use symbolic code and a translator program (ASM51)
- Example:

```
MOV A,P0
LCALL pwm_output
```

- This might be translated to...

Assembly code

- Solution: use symbolic code and a translator program (A51)
- Example:

```
0100 E5 80    MOV A,P0
0102 12 0253 LCALL pwm_output
```

- P0 is a *symbol* with a value of 80h
- 0253 is the value of label *pwm_output*

Assembly program example

```
            DSEG at 20h

cnt:        DS 1
blkst:      DS 10

            CSEG at 100h

start:      MOV cnt,#10     ; load cnt with 10
            MOV R0,#blkst
            CLR A
loop:       MOV @R0,A
            INC R0
            DJNZ cnt,loop
stop:       JMP STOP
            END
```

Assembler terminology

- DSEG is a *psuedo op*
- Some psuedo ops:
  - DSEG - define data segment (at location 20h)
  - CSEG - define code segment (at location 100h)
  - DS n - define n bytes of storage
  - END - end of assembly program
- Cnt, blkstr, start, loop, and stop are *labels*
- ; load … is a *comment*

Assembly code process

- Source code → A51 → Object code
- Two type of Object Code:
  - Absolute (all addresses specified)
  - Relocatable (addresses filled in later. NOT DONE!)
- Absolute Object code → DSW51 (dScope) or OBJ → OH51 (object to hex) → EPROM
- Relocatable OBJ + LIB → BL51 (linker) → dScope (or → OH51 → EPROM)

Intel Hex format

```
:0B0003 00 E4FF0FBF14FC7F14DFFE22 9F
:03 0000 00 02000E ED
:0C 000E 00 787FE4F6D8FD758107020003 3E
:00 0000 01 FF
```

- Byte count + start address + record type + data + checksum
- Checksum ED = -(03+02+0E)
- Question: What does this do for the first 100 microsec or so?

Disassembly example

```
0000 00 02000E ED
000E 00 787F E4 F6 D8FD 758107 020003

  0000 02 000E LJMP 000E
  000E 78 7F   MOV R0,#7f
  0010 E4      CLR A
  0011 F6      MOV @R0,A
  0012 D8 FD   DJNZ R0,-3 ;jmp 0011
  0014 75 8107 MOV SP, #7 ;sp=sfr 81
  0017 02 0003 LJMP 3
```