EdSim51 Tutorial

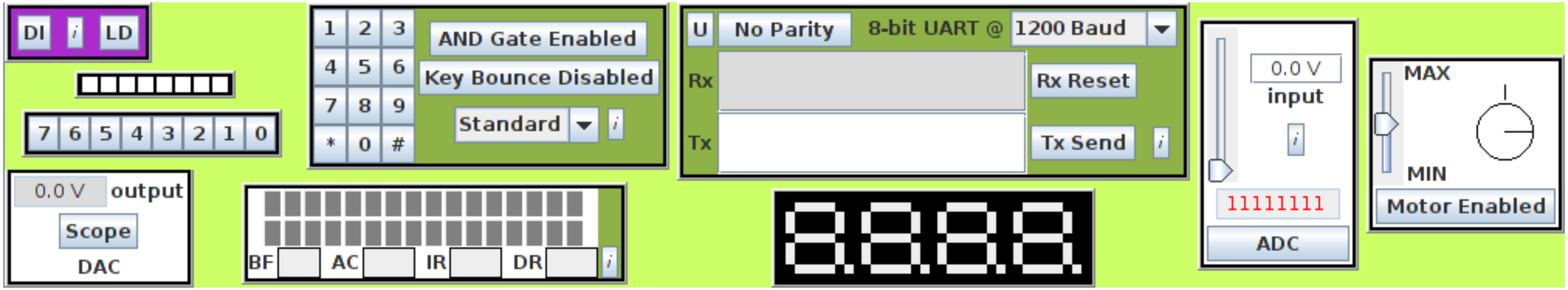
Pai H. Chou
National Tsing Hua University

EdSim51

- Download EdSim51
  - From http://edsim51.com/
  - Runs as a Java app
- Two versions
  - Edsim DI - standard, with everything
  - Edsim SH - customizable

EdSim51 DI

- DI = Dynamic Interface
  - Simulates a complete embedded system
  - LCD, LEDs, keypad, bank of buttons, ADC, DAC
  - cycle-accurate processor



AND Gate Enabled

Key Bounce Disabled

Standard

No Parity

8-bit UART @ 1200 Baud

0.0 V output

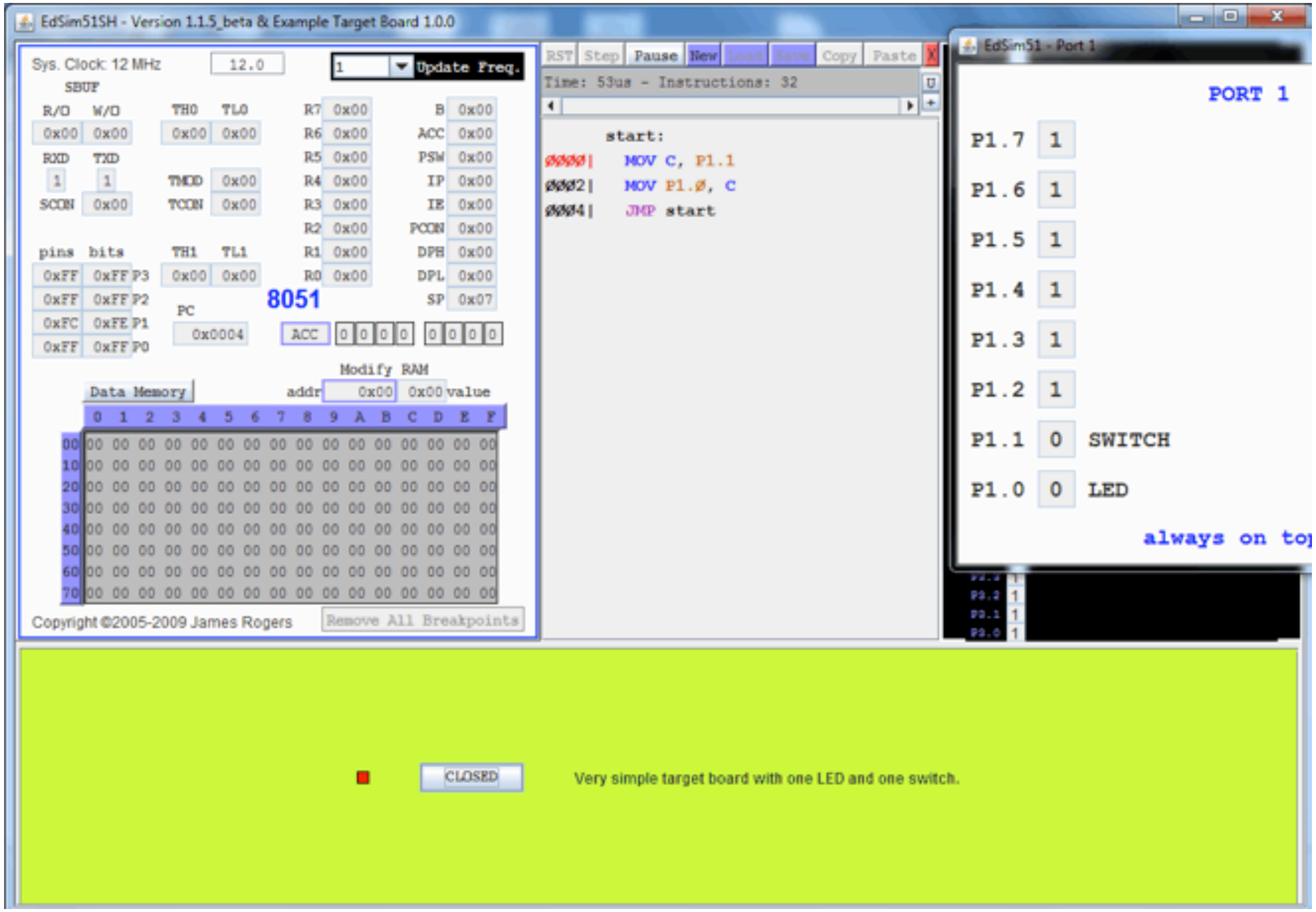
0.0 V input

11111111

Motor Enabled

EdSim51 SH

- Customize the devices to use



EdSim51SH - Version 1.1.5_beta & Example Target Board 1.0.0

Sys. Clock: 12 MHz 12.0 | 1 Update Freq.

Time: 53us - Instructions: 32

```
        start:
ØØØØ|   MOV C, P1.1
ØØØ2|   MOV P1.Ø, C
ØØØ4|   JMP start
```

SBUF R/O 0x00 W/O 0x00

RXD 1 TXD 1

SCON 0x00

TH0 0x00 TL0 0x00

TMOD 0x00

TCON 0x00

TH1 0x00 TL1 0x00

pins	bits	
0xFF	0xFF	P3
0xFF	0xFF	P2
0xFC	0xFE	P1
0xFF	0xFF	P0

PC 0x0004

R7	0x00	B	0x00
R6	0x00	ACC	0x00
R5	0x00	PSW	0x00
R4	0x00	IP	0x00
R3	0x00	IE	0x00
R2	0x00	PCON	0x00
R1	0x00	DPH	0x00
R0	0x00	DPL	0x00
		SP	0x07

8051

ACC 0 0 0 0 0 0 0 0

Modify RAM addr 0x00 0x00 value

Data Memory

```
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Copyright ©2005-2009 James Rogers

EdSim51 - Port 1

PORT 1

P1.7	1	
P1.6	1	
P1.5	1	
P1.4	1	
P1.3	1	
P1.2	1	
P1.1	0	SWITCH
P1.0	0	LED

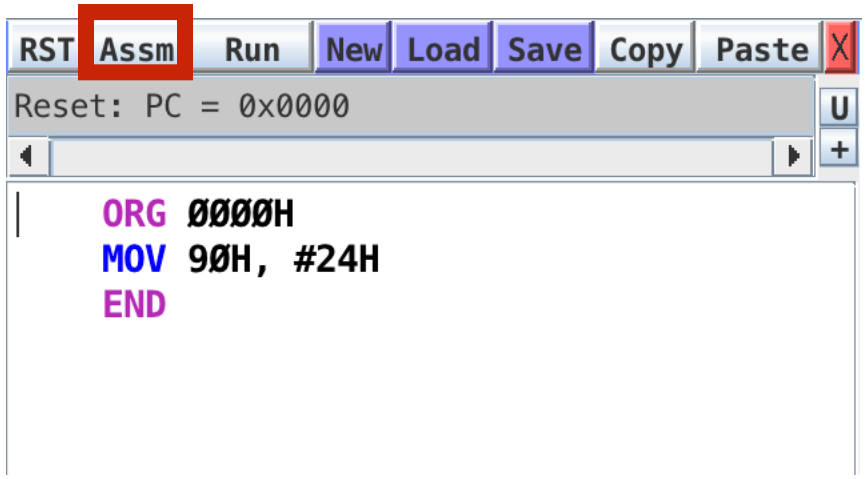
always on to

P3.2 1
P3.1 1
P3.0 1

CLOSED

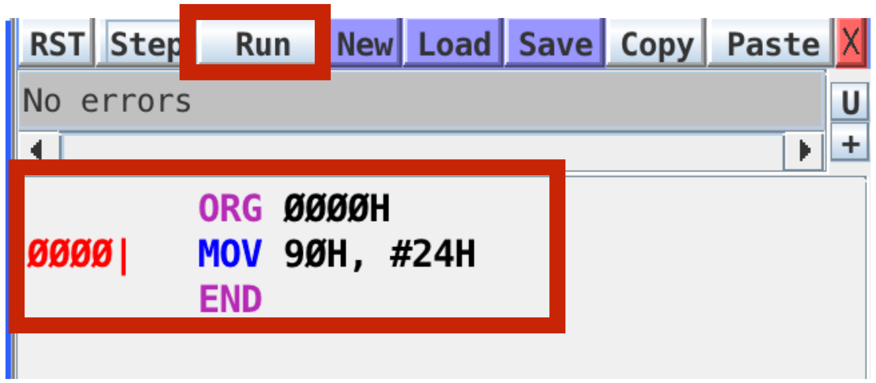
Very simple target board with one LED and one switch.

First Program on bare metal



```
Reset: PC = 0x0000
```

```
    ORG 0000H
    MOV 90H, #24H
    END
```



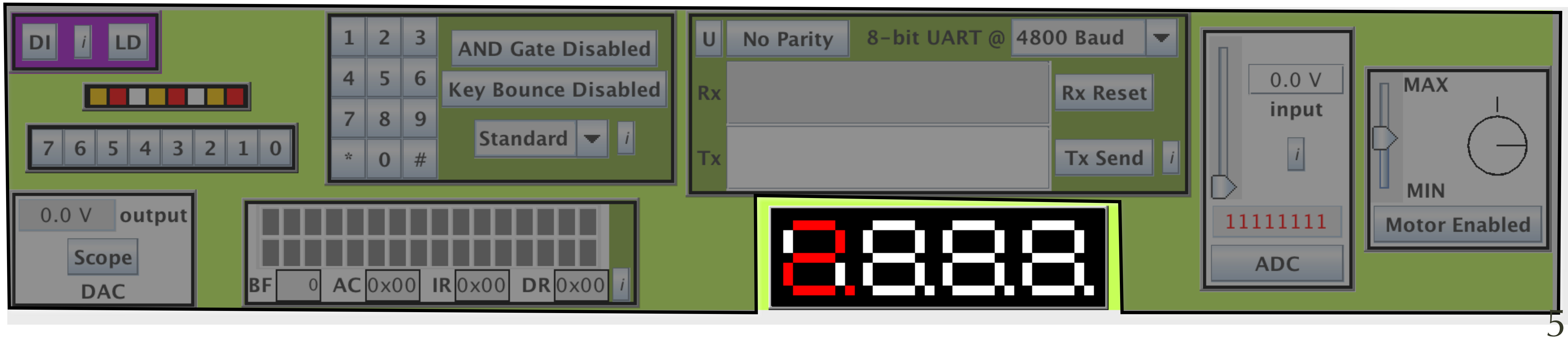
```
No errors
```

```
            ORG 0000H
0000|       MOV 90H, #24H
            END
```

(1) type these lines
(2) click Assm

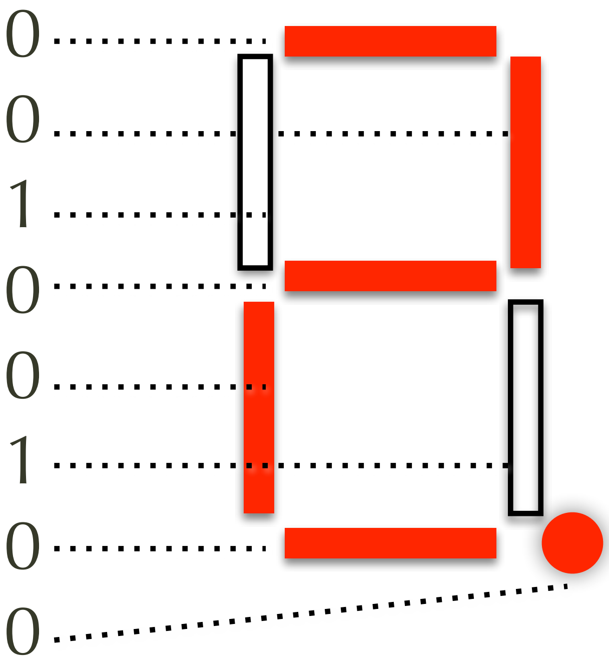
(3) should get
(4) click Run

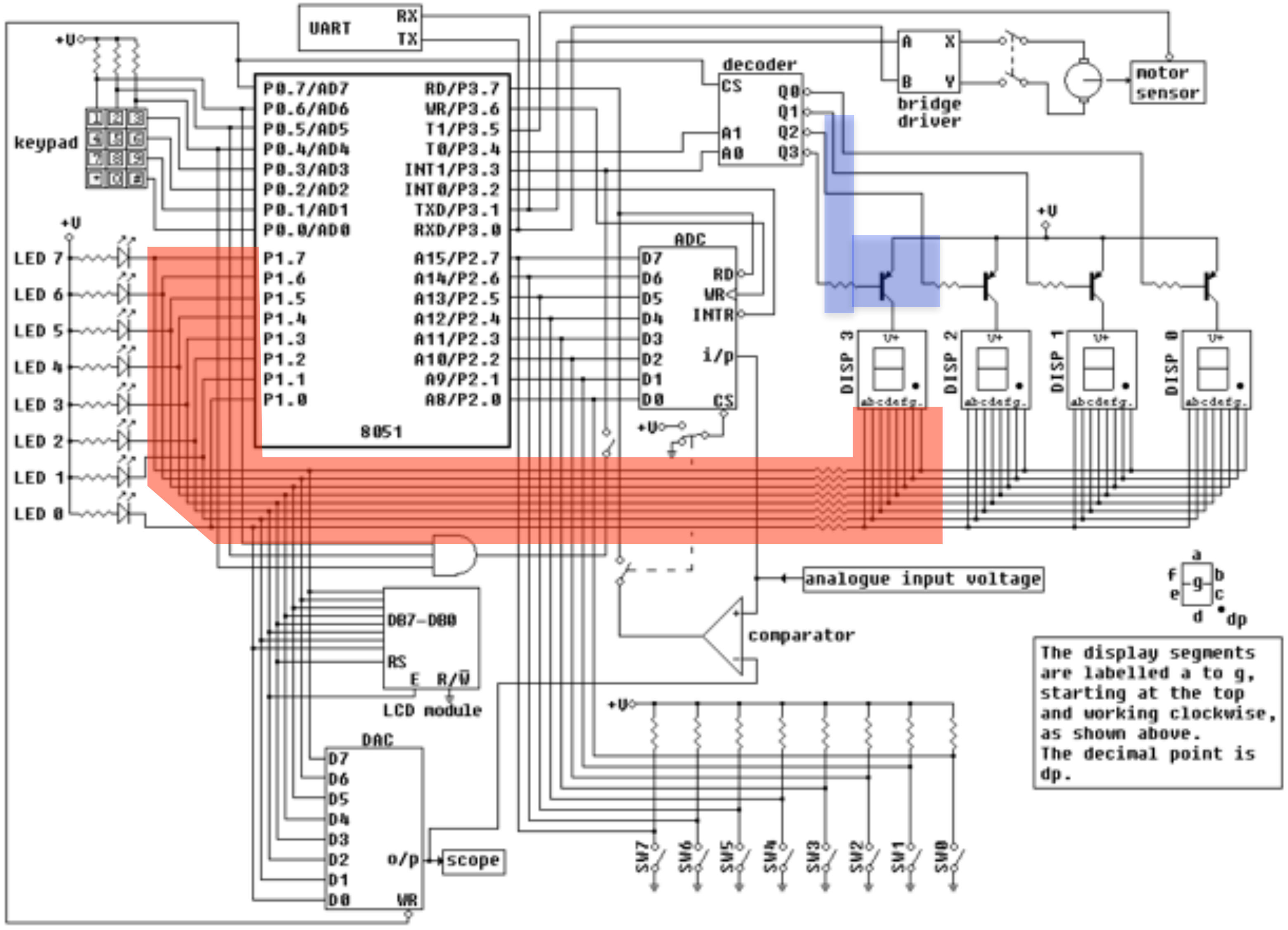
(5) LED should display "2."



How it works

- Port P1 (address 90H) gets 24H = 0010_0100b
- 0 is a pull-down => turns on!
- 1 is leaves it as pull-up => turns off LED!

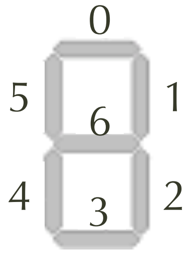


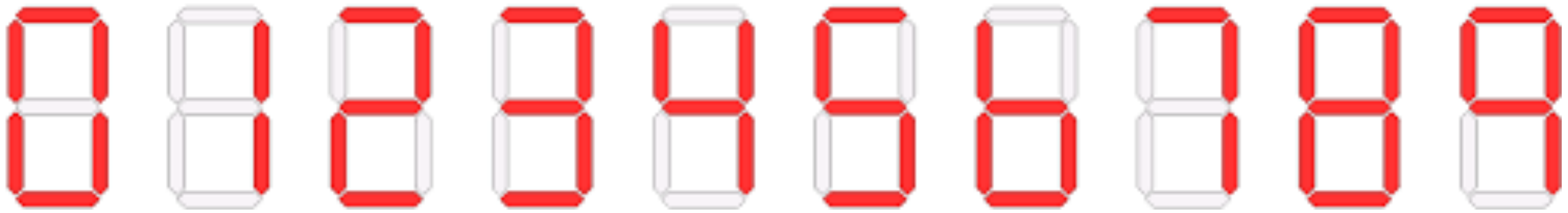


The display segments are labelled a to g, starting at the top and working clockwise, as shown above. The decimal point is dp.

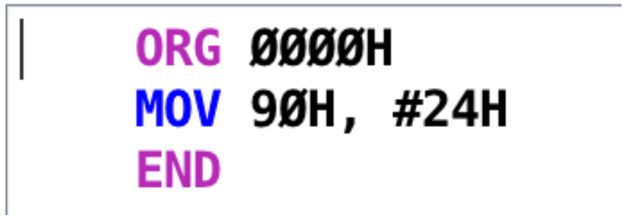
Try it yourself

- How to get it to display different digits, plus decimal point? example,





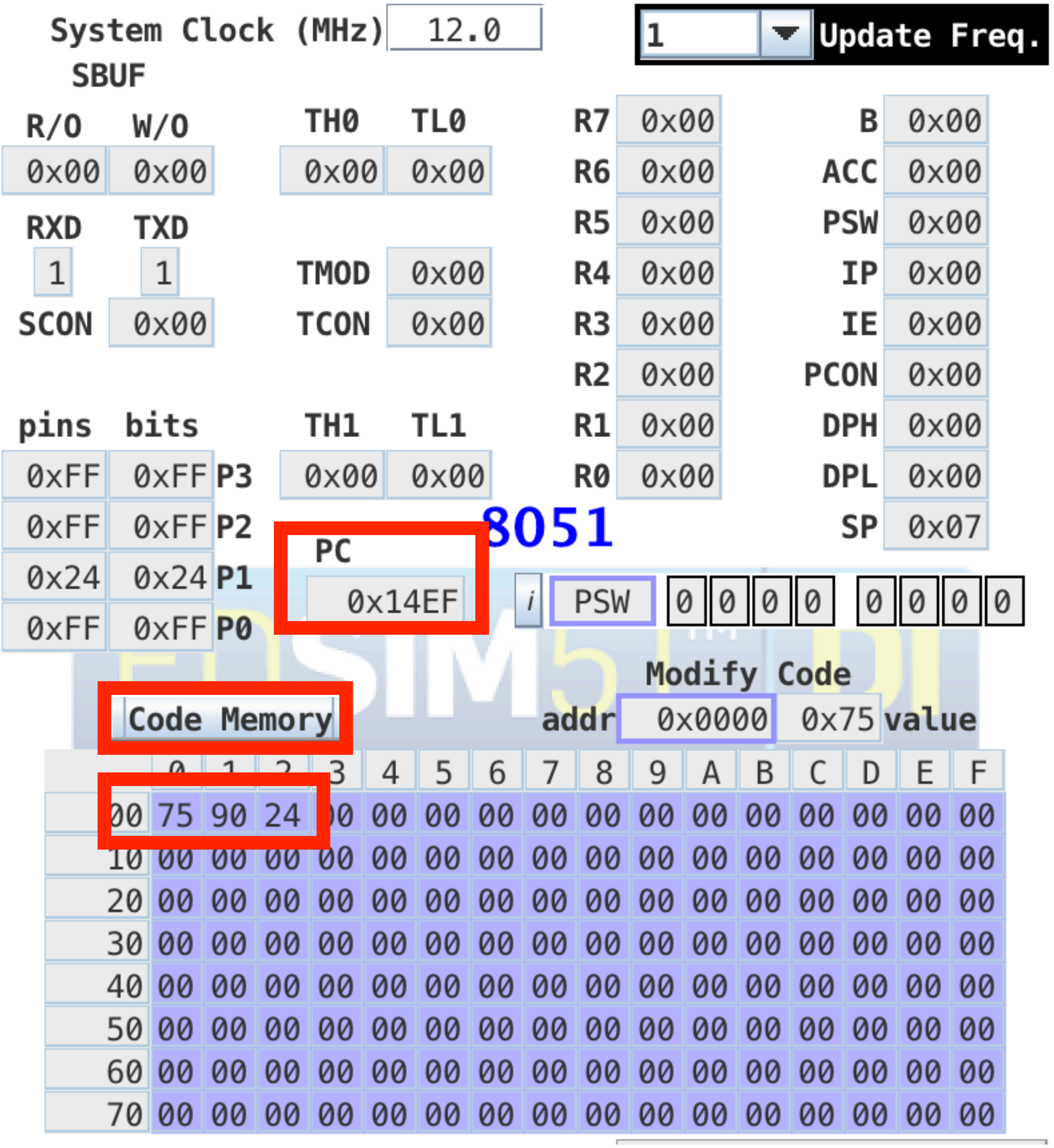
-



```
ORG 0000H
MOV 90H, #24H
END
```

Question

- What does END mean in assembly?
  - END is an "assembler directive"
  - It just means end of source code listing
  - it does not mean CPU stops running!
- Actually, CPU continues running
  - What instructions does it execute?



System Clock (MHz) 12.0

1 ▼ Update Freq.

SBUF

R/O	W/O
0x00	0x00

RXD	TXD
1	1

SCON 0x00

TH0	TL0
0x00	0x00

TMOD 0x00

TCON 0x00

TH1	TL1
0x00	0x00

pins	bits	
0xFF	0xFF	P3
0xFF	0xFF	P2
0x24	0x24	P1
0xFF	0xFF	P0

R7	0x00
R6	0x00
R5	0x00
R4	0x00
R3	0x00
R2	0x00
R1	0x00
R0	0x00

B	0x00
ACC	0x00
PSW	0x00
IP	0x00
IE	0x00
PCON	0x00
DPH	0x00
DPL	0x00
SP	0x07

8051

PC 0x14EF

i PSW 0 0 0 0 0 0 0 0

Modify Code

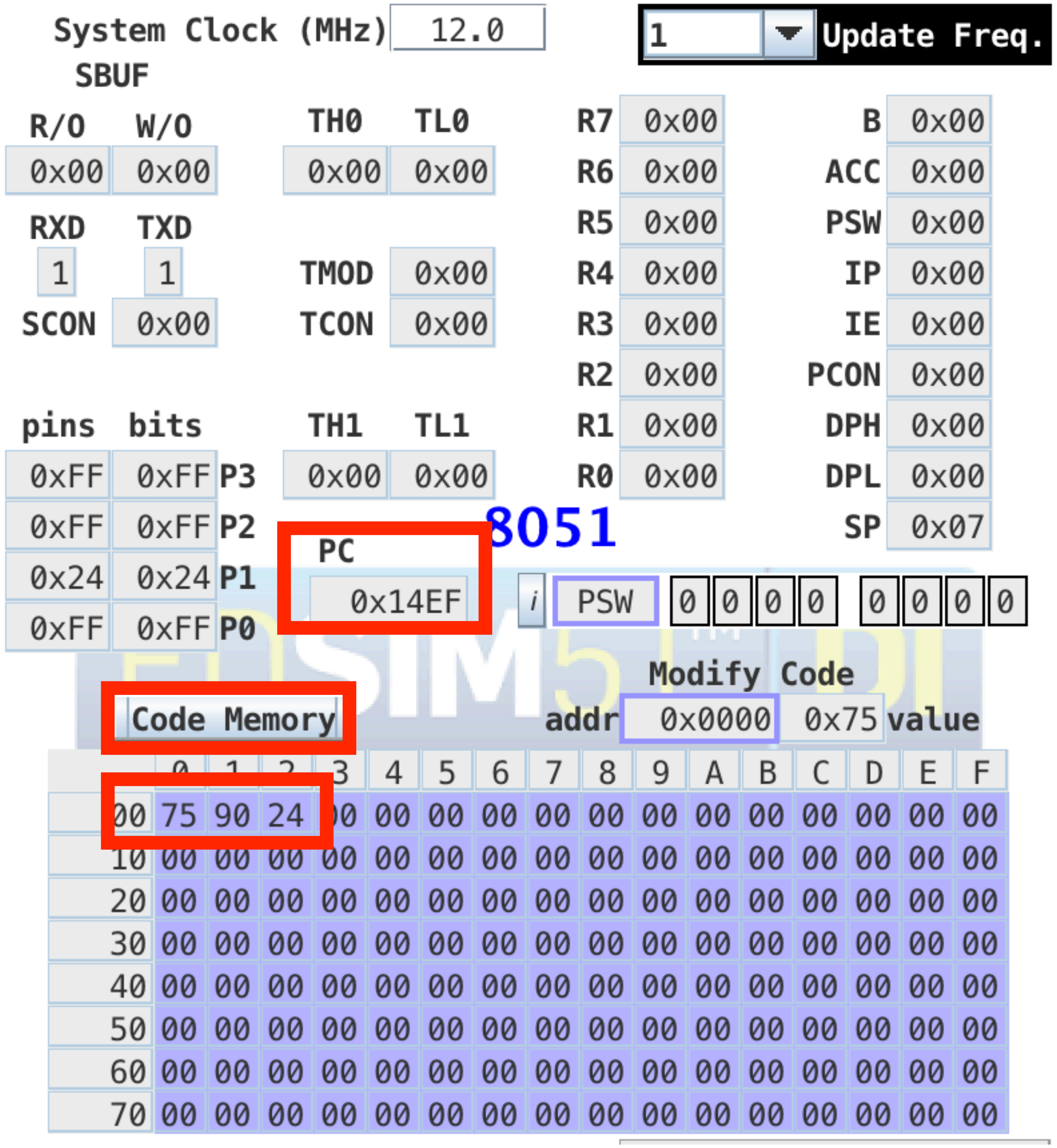
addr 0x0000 0x75 value

Code Memory

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	75	90	24	00	00	00	00	00	00	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Program execution

- Click on "Code Memory" or "Data Memory" button to toggle display of memory content
- PC 0x14EF is where in program the CPU is executing (you may see something else)
    - PC keeps incrementing until 0xFFFF, then wraps around to 0x0000
- Code memory contains all 00 except first three bytes
    - machine code 00 is the NOP instruction, means "do nothing"



System Clock (MHz) 12.0

1 ▼ Update Freq.

SBUF

R/O	W/O
0x00	0x00

RXD	TXD
1	1

SCON 0x00

TH0	TL0
0x00	0x00

TMOD 0x00

TCON 0x00

TH1	TL1
0x00	0x00

pins	bits	
0xFF	0xFF	P3
0xFF	0xFF	P2
0x24	0x24	P1
0xFF	0xFF	P0

R7	0x00
R6	0x00
R5	0x00
R4	0x00
R3	0x00
R2	0x00
R1	0x00
R0	0x00

B	0x00
ACC	0x00
PSW	0x00
IP	0x00
IE	0x00
PCON	0x00
DPH	0x00
DPL	0x00
SP	0x07

8051

PC 0x14EF

i PSW 0 0 0 0 0 0 0 0

Modify Code

addr 0x0000 0x75 value

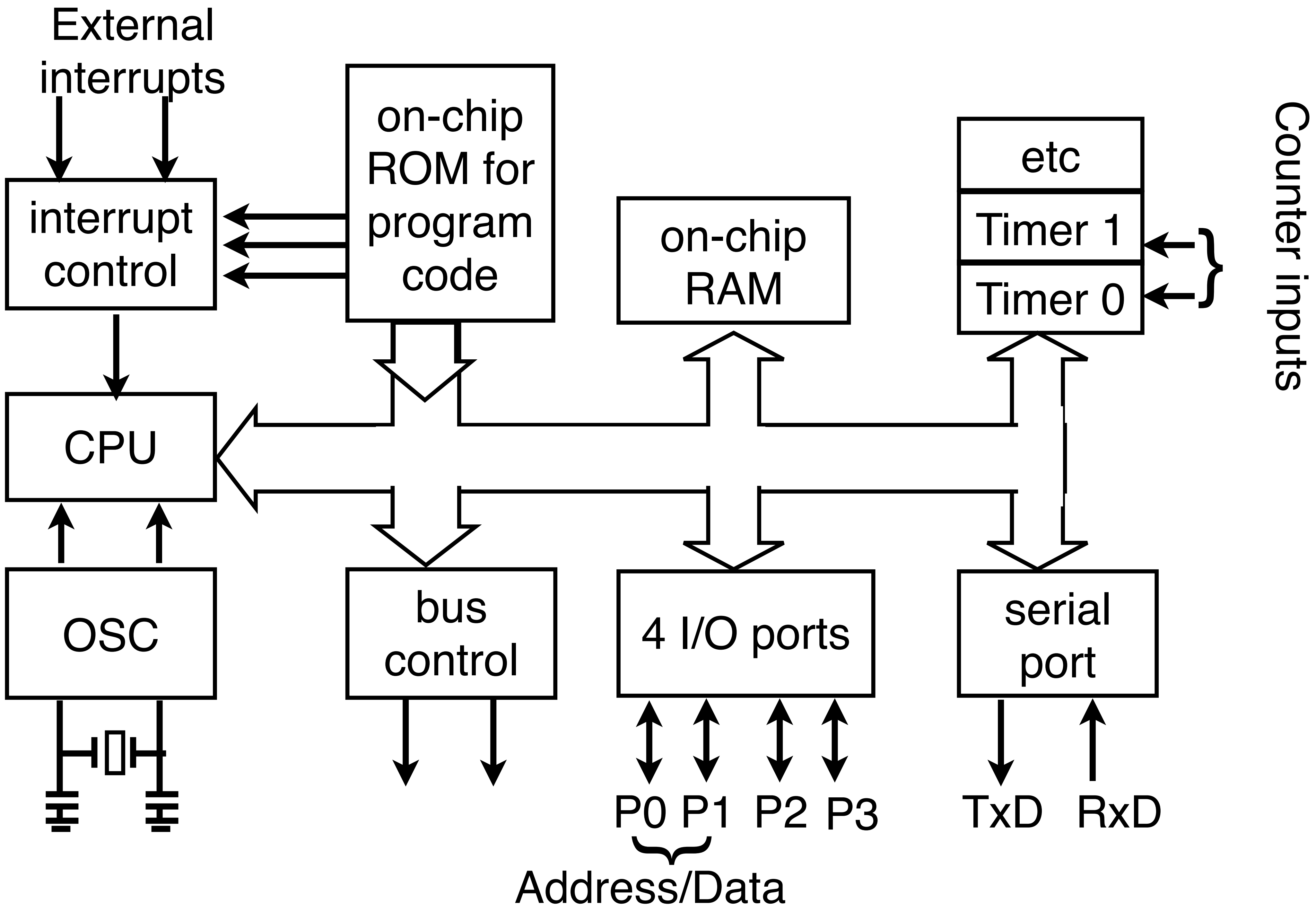
Code Memory

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	75	90	24	00	00	00	00	00	00	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Processor in EdSim51

- The Intel 8051 (MCS-51) microcontroller
  - http://lms.nthu.edu.tw/sys/read_attach.php?id=414787
- 8-bit words
- 16-bit address (external), 8-bit address (internal)
- Harvard architecture
  - 64KB "external" data memory, 256-byte "internal" mem
  - separate 64 KB code memory

Block Diagram of 8051



Memory Spaces in 8051

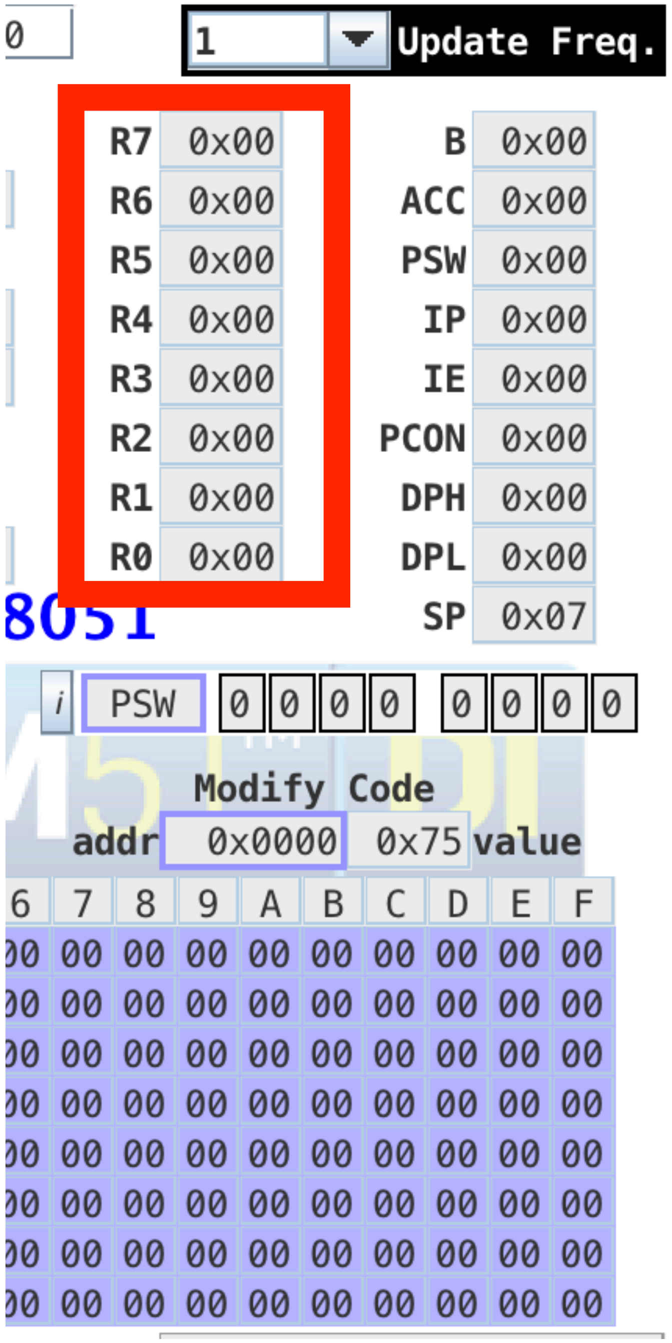
Space	CODE	IDATA	XDATA
full name	program memory	internal data memory	external data memory
Size	64 KB	256 Bytes (not KB)	64 KB
Purposes	instruction and constant data	CPU registers, hardware stack, small variables	software stack, main memory

Registers in 8051

- General purpose, 8-bit
  - A: (Accumulator), B
  - R0, R1, ..., R7 (CPU registers, in 4 banks)
- 16-bit, specifically used as pointers
  - DPTR: data pointer, concatenated DPH,DPL
  - PC: (program counter) not user visible
- PSW: program status word (8-bit)

Banks of CPU Registers

- One set of 8 registers visible at a time
  - R0, ... R7 => selected using 3 bits
- Four banks of CPU registers, in IDATA
  - bank 0: IDATA addresses 0x00-0x07
  - bank 1: IDATA addresses 0x08-0x0F
  - bank 2: IDATA addresses 0x10-0x17
  - bank 3: IDATA addresses 0x18-0x1F
  - bank selected by setting a special function register



1

R7	0x00	B	0x00
R6	0x00	ACC	0x00
R5	0x00	PSW	0x00
R4	0x00	IP	0x00
R3	0x00	IE	0x00
R2	0x00	PCON	0x00
R1	0x00	DPH	0x00
R0	0x00	DPL	0x00
		SP	0x07

PSW 0 0 0 0 0 0 0 0

addr 0x0000 0x75 value

```
6  7  8  9  A  B  C  D  E  F
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
```

Accumulator (A)

- An implicit register in many instructions
  - as both a source and the destination
    e.g, ADD A, #23
    meaning: A = A + 23
- Reason for using A
  - small code size, because there is just one!
  - All others require several bits for registers

Machine Instructions

- Opcode
  - Specifies the operation (~function)
- Operands
  - the "arguments" to an opcode
  - could be accumulator, register, constant value, value in memory, etc

Opcodes in 8051

- MOV, MOVX, MOVC, XCH, XCHD, PUSH, POP
- ADD, ADDC, SUBB, MUL, DIV, ANL, ORL, XRL
- RR, RL, RLC, RRC, SWAP
- INC, DEC, CLR, SETB, CPL, DA
- NOP
- AJMP, LJMP, ACALL, LCALL, RET
- JB, JNB, JC, JNC, JZ, JNZ, JMP, CJNE, DJNZ

Idiosyncrasy with immediate in Intel Assembly syntax

- Default base: decimal
  - #12 (assumed to be decimal)
  - Can be hex: #12**H** (12 hex, = 18 dec.)
- However! the char after # must be 0..9
  - #FF**H** is not an immediate (since F is not in 0..9)
  - Solution: #**0**FF**H** (add a useless 0 (zero) in front. It does not make it octal)

Immediate vs. direct (Addressing mode)

- MOV A, #17H ;; #17H is a *literal value*
  meaning: A = 0x17;
- MOV A, 17H ;; 17H is IDATA address!
  meaning: A = *((char*)0x17);
- Big difference!
  - R0, ... R7 => register mode
  - #17 => immediate mode;
  - 17H => direct mode (IDATA address 0x17)
  - 17 => direct mode at decimal 17 (instead of hex)

MOV instruction

- syntax:
  MOV dest, src
  - Think assignment statement: dest := src;
- dest, src are called Operands
  - dest can be A, B, R0..R7, DPH, DPL
  - src can be A, B, R0...R7, or an immediate
- *Immediate* is aka a "constant", "literal" value, e.g., #12

Allowed Combinations of byte-Addressing Modes

Opcode	Dest	Source
MOV	A,	Ri or @Ri
		#imm
		dir
	Ri, @Ri,	A
		#imm
		dir
	dir,	A
		#imm
		dir
		Ri or @Ri

Note: @Ri is limited to @R0 or @R1

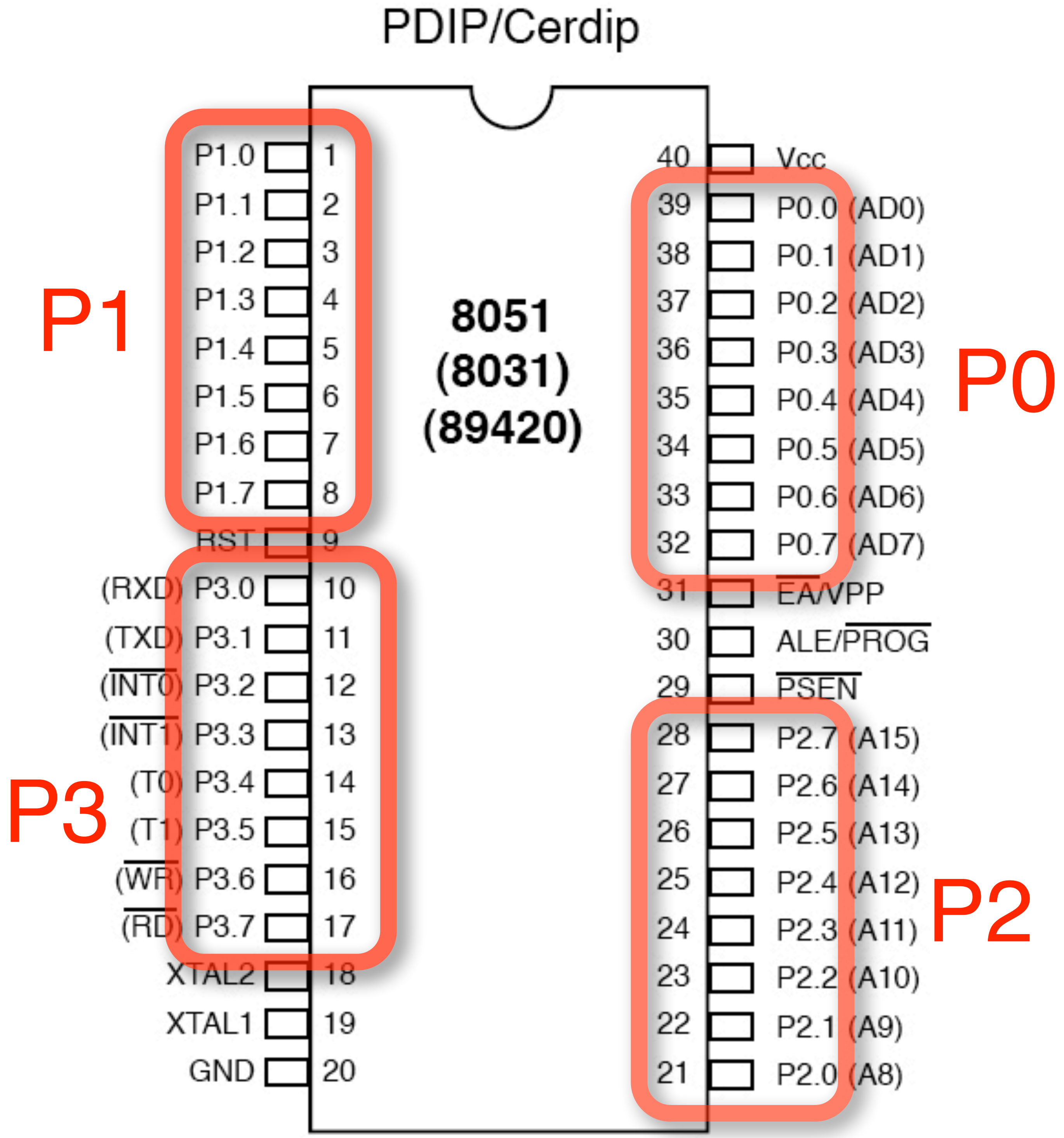
Ri can be from R0 .. R7

Restricted combinations of Addressing Modes

- Disallowed: Register-to-register MOV
  - e.g., MOV ~~R1, R2~~
  - solution: go through A or use immediate
- Accumulator-to-accumulator MOV ~~A, A~~ (useless)
- anything-to-immediate MOV (nonsense)
  - e.g., MOV ~~#20, R3~~

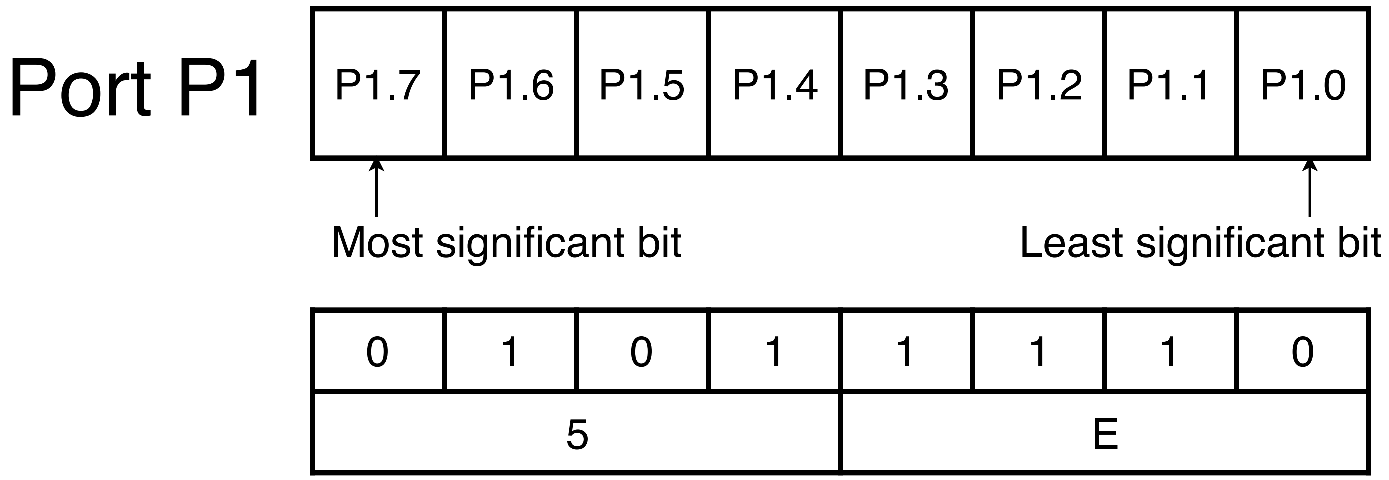
8051 ISA: Four I/O ports

- 8-bits each
  P0, P1, P2, P3
- Direct addresses
  80H, 90H, A0H, B0H
- Difference: values tied to the pins
- Bit addressable



Output: write to port latch

- Byte access
  - MOV   P1, #5EH
  - big-endian bit order
- Bit access
  - SETB   P1.1 *;; sets port 1 bit 1*
  - CLR   P2.3 *;; clears port 2 bit 3*



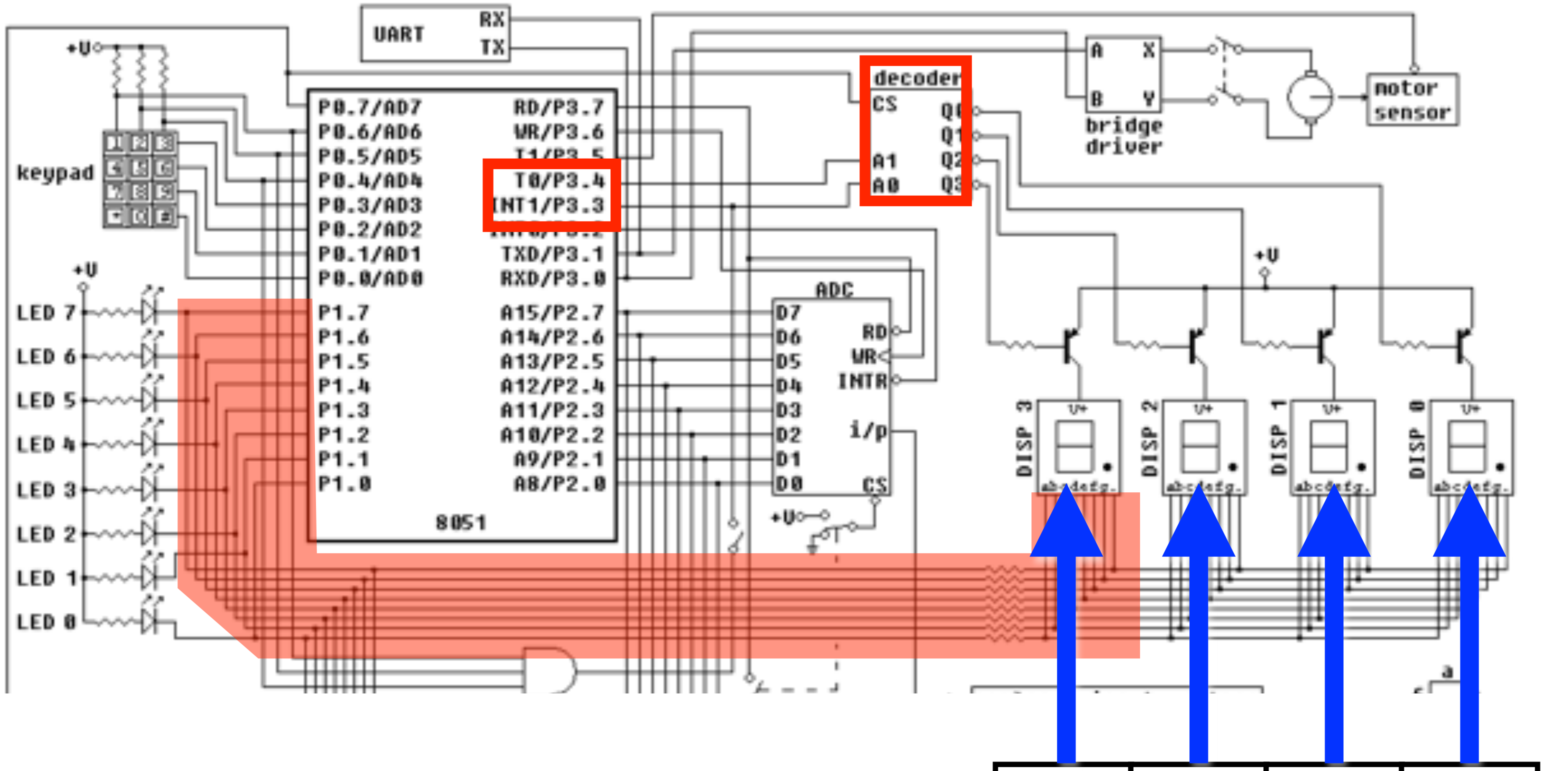
Port P1

P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
0	1	0	1	1	1	1	0
5				E			

Most significant bit

Least significant bit

Example: how to light up another 7-segment LED



select by setting <P3.4,P3.3> to...

11	10	01	00

- Need to select digit
  - Decoder maps <A1,A0> to one-hot
  - Controlled by <P3.4,P3.3>
- SETB or CLR instruction to assign = 1 or 0

How to light up all four digits?

- Answer: need to continue refreshing
- Example, want to write "2019"
  - select digit 3, display digit "2"
  - select digit 2, display digit "0"
  - select digit 1, display digit "1"
  - select digit 0, display digit "9"
  - repeat! (can use SJMP instruction)

What does this code do?

```
            ORG     0
TOP:        SETB    P3.4
            SETB    P3.3
            MOV     P1, #24H
            CLR     P3.3
            MOV     P1, #24H
            CLR     P3.4
            SETB    P3.3
            MOV     P1, #24H
            CLR     P3.3
            MOV     P1, #24H
            SJMP    TOP         ;; jump to TOP
            END
```

Try it yourself

- Try to print out "2019" in an infinite loop