CPE 212 [illegible] 2003

(A) Why am I taking this course?

- Microcontrollers (especially 8051 family) are used to design many embedded systems such as automotive sys. network cards, telephone, etc...

(A) 8051 family (developed by Intel.co)

- classic (> 15 years)
- most popular
  - many applications
  - many peripherals available
  - much development software available
- more than 150 variants of 8051 offered by more than 20 different vendors.
- > 126 million components sold annually

We will learn about what's inside, how to program, and how to design hardware around

These concepts that we will learn are fundamental to digital systems design, in general.

(1) Quick Review on digital-logic

Systems:

1. Binary (base 2): one binary digit => bit
   4 " s => nibble
   8 " s => byte

ex) 0010 1010_2

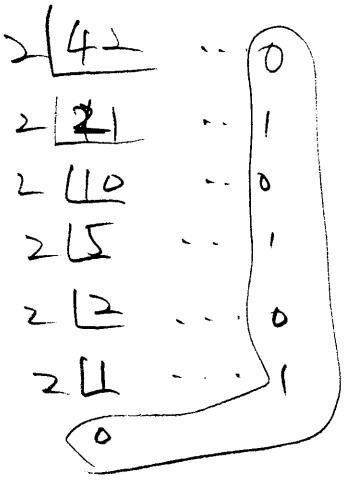
- has 8 bits.
- has 2 nibbles
- has 1 byte.

Conversion to decimal (base 10).

- Consider weights of each digit.

Bin: 0 0 1 0 1010_2 = $32 + 8 + 2 = 42_{10}$.
dec: 128 64 (32) 16 (8) 4 (2) 1

Decimal to binary? ⇒ successive division algorithm.
(continuously divide by two)



⇒ 0010 1010_2

Octal (base 8).

3 binary digits = 1 octal digit.

= 0010 1010_2

0 5 2 = 52_8.

⭐ Hexadecimal #. (base 16)

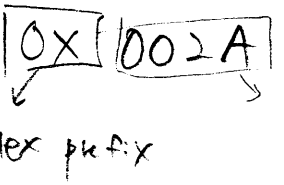
4 binary digits = 1 nibble = 1 ~~hex~~ digit.

ex) $\underbrace{0010}\ \underbrace{1010}_2$

$2 \quad A_{16}$

If the default system data word size is 16 bits.. (= 4 Hex digits)

ex) in C programing language

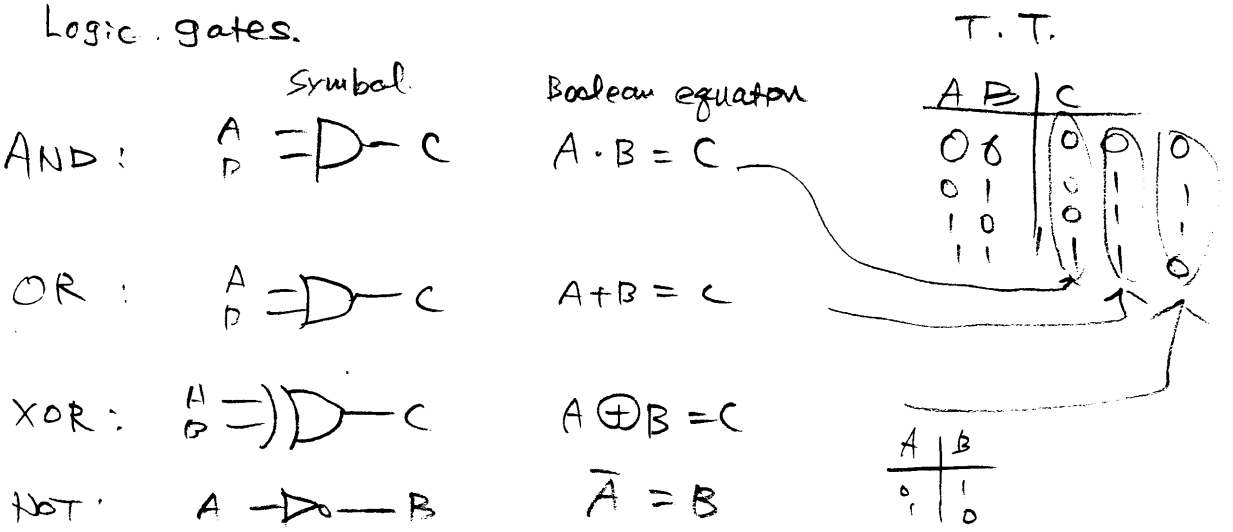


in ASM (Intel-compatible assembler?

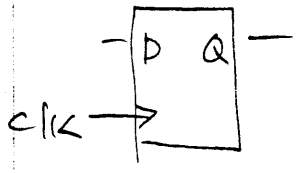
002A H

Hex postfix.

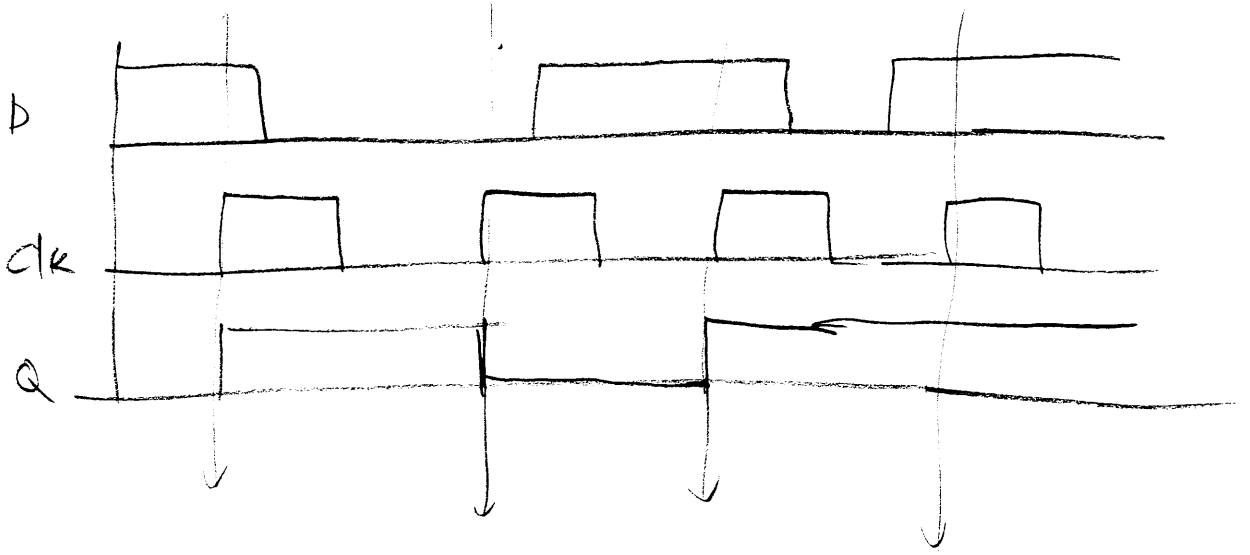
⭐ Logic gates.



④ D Q flip-flop. (positive-edge-triggered)

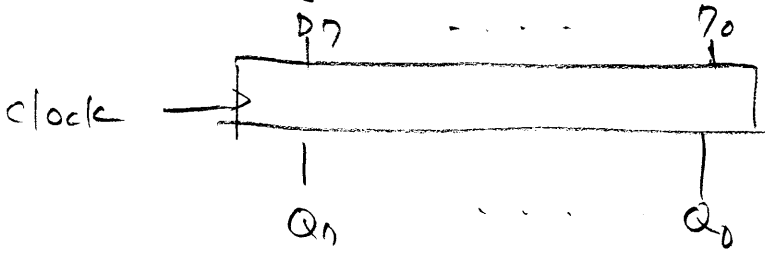


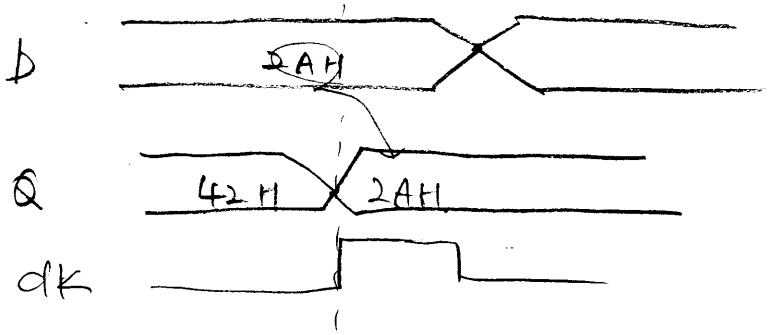
D(t)	Q(t+1)
0	0
1	1



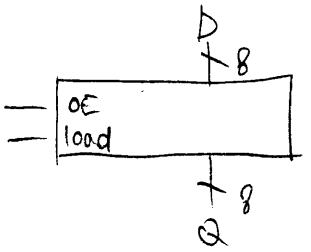
⑤ Register : a logic element to store an n-bit binary word.

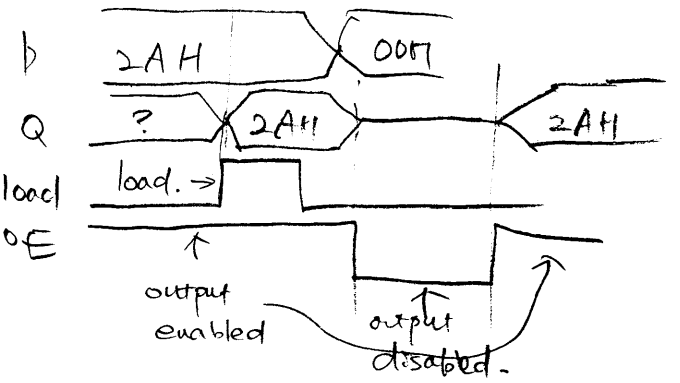
e.g) 8 bit register



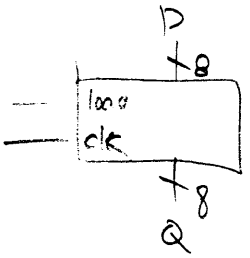


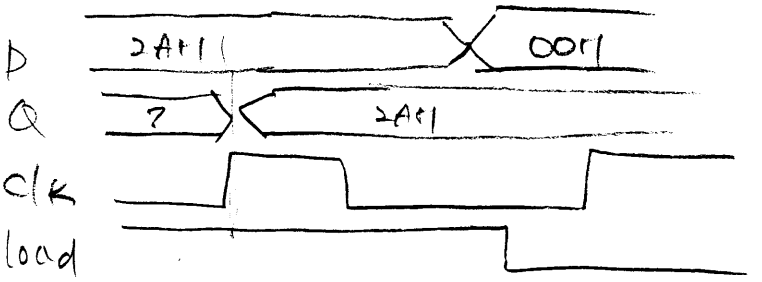
type 1 register with OE (output enable) and load (load register) signals.



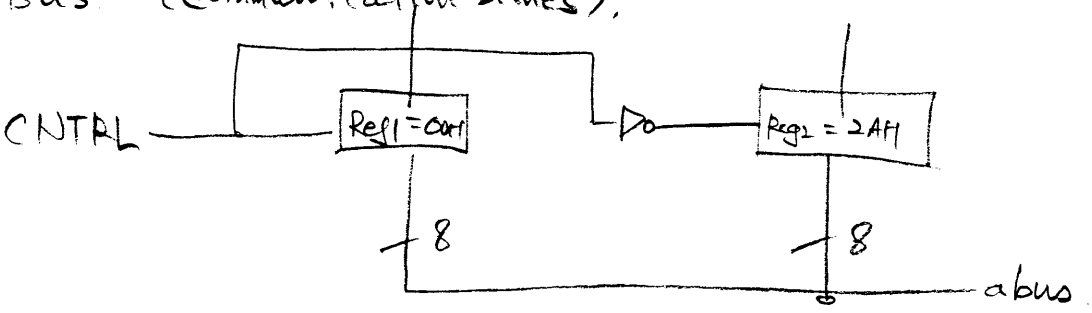


type 2 register with load & clk.





Bus (communication lines).

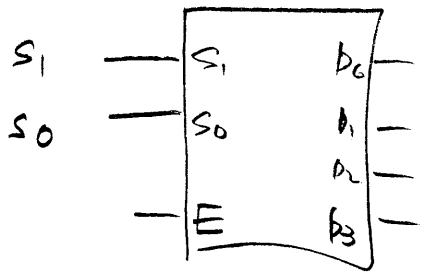


CNTRL	abus
0	2AH
1	00H.

⇒ allows devices to share bus!

★ Decoder

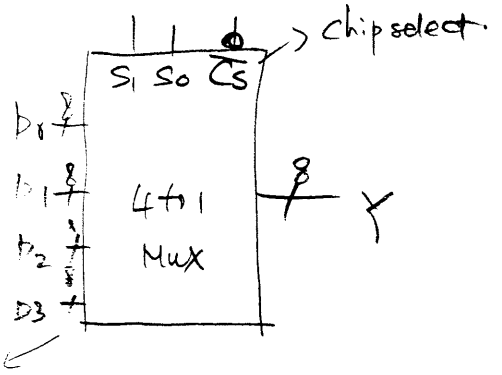
2 to 4 decoder



$\bar{E}$	S_1	S_0	D_0	D_1	D_2	D_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	X	X	0	0	0	0

~~handout & explain!~~

★ Mux

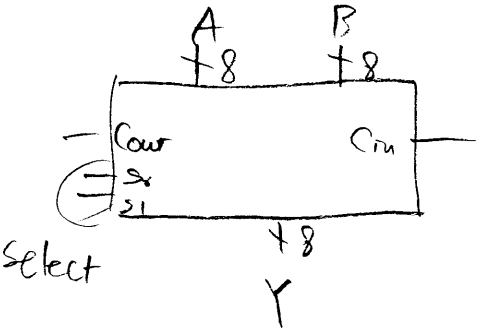


also

$\overline{CS}$	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	X	X	High Impedance

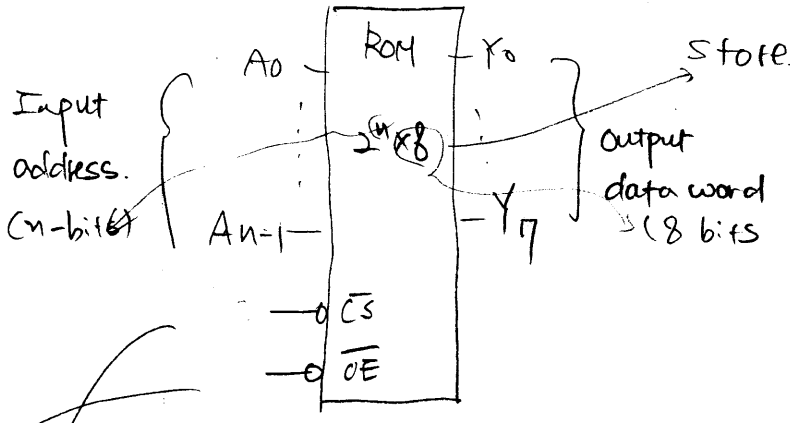
★ ALU

ex)



S_0	S_1	OP
0	0	ADD
0	1	SUB
1	0	AND
1	1	OR

ROM permanent storage of data



stores 2^n 8-bit datums.

ex) 4K x 8 for n = 12.

chip select. if $\overline{OE} = 0$, then the ROM is enabled.

Output Enable. If $\overline{CS} = 0$, then the output word becomes available

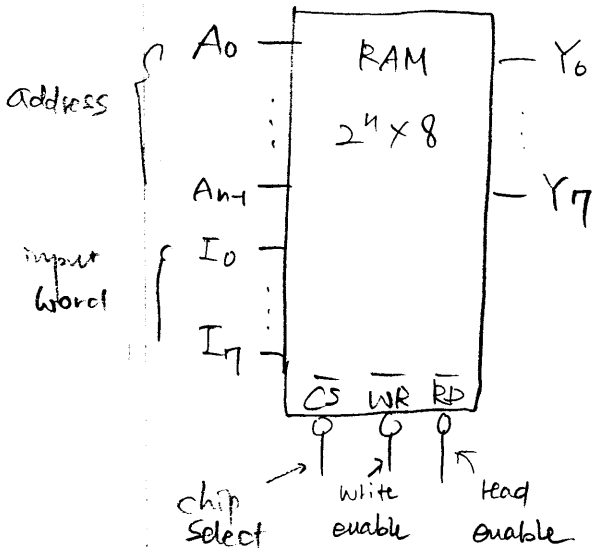
ex)

Address	data word
000H	2AH
001H	42H
002H	00H
:	:
FFFH	FAH.

① A = FFFH, $\overline{OE} = 0$ and $\overline{CS} = 0$ ⇒ Y = FAH.

② $\overline{OE} = 1$ or $\overline{CS} = 1$, then Y = high impedance.

RAM



$\overline{CS}$	$\overline{WR}$	$\overline{RD}$	
1	x	x	Do nothing Y = high impedance
0	0	1	write I to address location A. Y = high impedance
0	1	0	Y = value at loc A.
0	0	0	not used
0	1	1	do nothing, Y = high imp

⭐ Memory Characteristics.

⭐ Registers – small, fast, temporary storage and usually on-chip

⭐ RAM – large, slow (↘ than register), volatile, on or off chip.

⭐ ROM – non-volatile, slow (↑ than register), large mem.

types: EPROM (Electrically programmable ROM)
can be erased (entirely) with UV light
fast read.

EEPROM (Elec Erasable PROM)
ex. write – simple addr electrically.

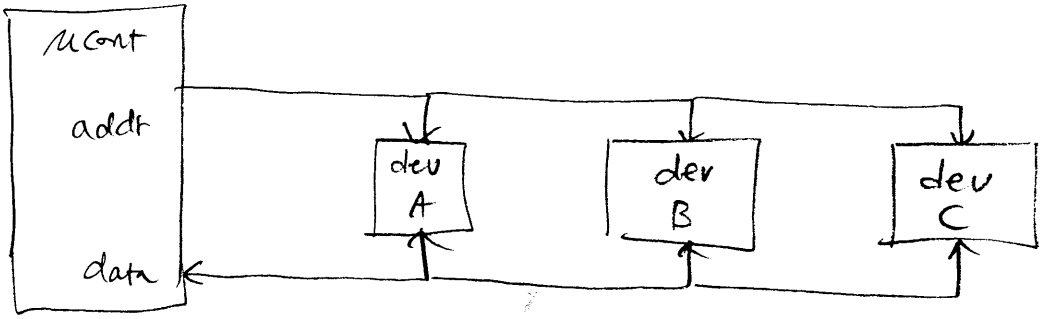
FLASH

block-wise erase (one or more blocks erased at a time)

Address decoding : Allows μCont / μ proc to "talk" with only one device at a time.

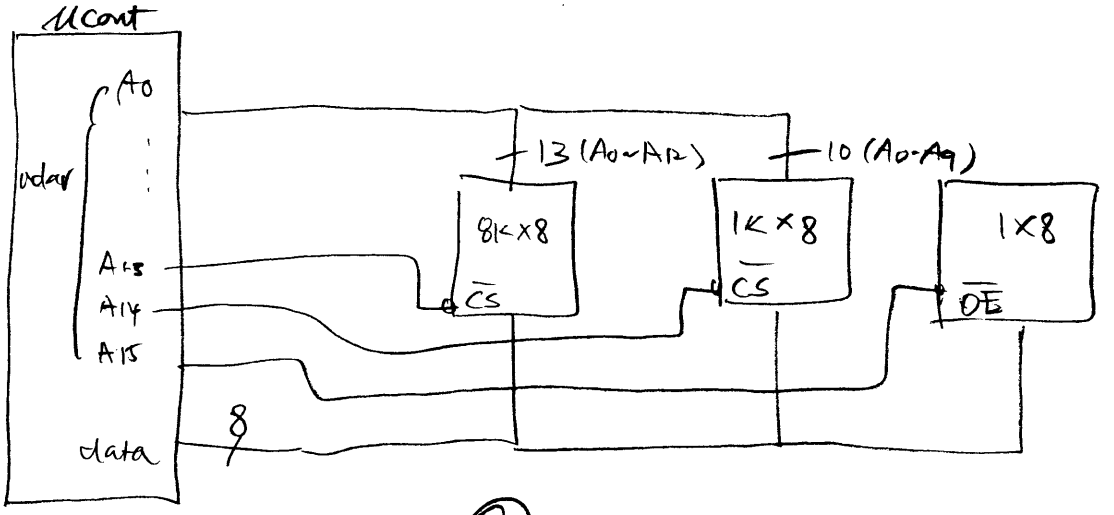
A μ cont often needs to talk to several devices through a single port (or bus)

ex)



How? ⇒ Address decoding.

Method 1 : Direct connection



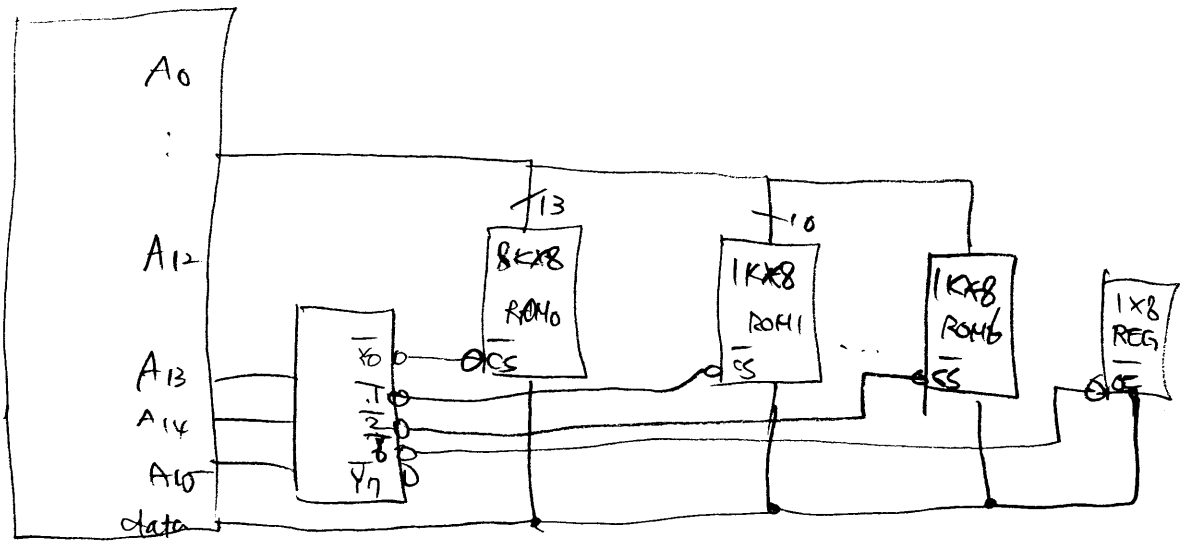
Since $2^{13} = 1024 \times 8 = 8K$

Device	enabling addr $A_{15}\ A_{14}\ A_{13}\ A_{12} \cdots A_0$	addr space.	used addr.
8K X8	1 1 0 X ··· X	0XC000 ~ 0X DFFF	8K
1K X8	1 0 1 X ··· X	0XA000 ~ 0X BFFF	8K
1X8	0 1 1 X ··· X	0X6000 ~ 0X 7FFF	8K
not used.	0 0 0		
	0 0 1		
	0 1 0		
	1 0 0		

We need $8K + 1K + 1 \approx 9K$

Used $8K \times 3 = 24K \Leftarrow \approx 15K$ addr space wasted.

Method 2: decoder (advanced & more efficient method)

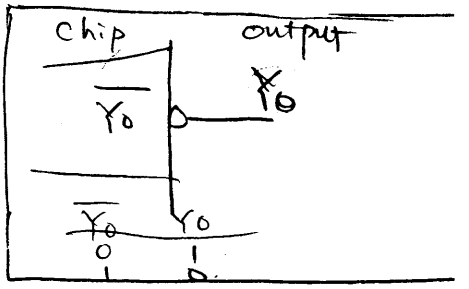


Size(B)	Device	addrs	addr space	used
8K	ROM1	000 X···X	0X0000 ~ 0X1FFF	8K
1K	2	001 X···X	0X2000 ~ 0X3FFF	8K
⋮	⋮	⋮	⋮	
1K	16	110 X···X		
1	REG	111 X···X	0XE000 ~ 0XFFFF	8K

ex) 74 LS 138. (typo. 1st page $\overline{OE}$ → $\overline{CS}$ for RAMs)
handout

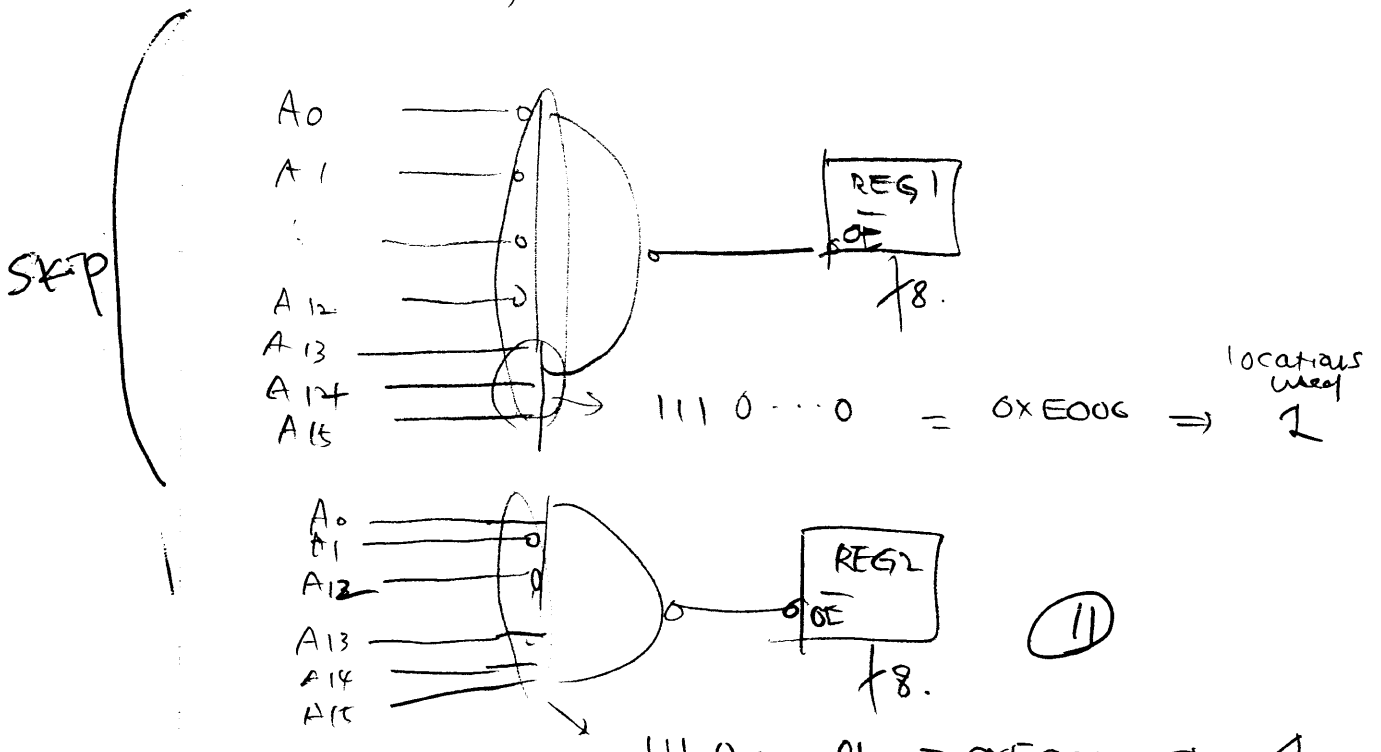
① A=0 B=0 C=0 (if $\overline{G2A} \cdot \overline{G2B} \cdot G1$)

⇒ RAM 0 enabled



① 111 ⇒ REG enabled.

method 3 : Customized decoding circuit or PLA
(very precise but costly)



Ⓐ µPs vs. µCs.

µPs: single-chip CPUs used in µcomputers.

µCs: performing "control" functions by interfacing with the "real world" to turn devices on & off and to control conditions.

Ⓑ 3 main differences.

① HW architecture

µP is a single chip CPU.
µC contains µp, RAM, ROM, I/O interface, timer interrupt scheduling circuitary, etc...

② Applications.

µPs for Arithmetic & logic ops. (information processing)
µCs for "control" I/O devices in real time

③ Inst set features.

µPs ⇒ processing intensive Inst set.
µCs ⇒ control intensive inst set.

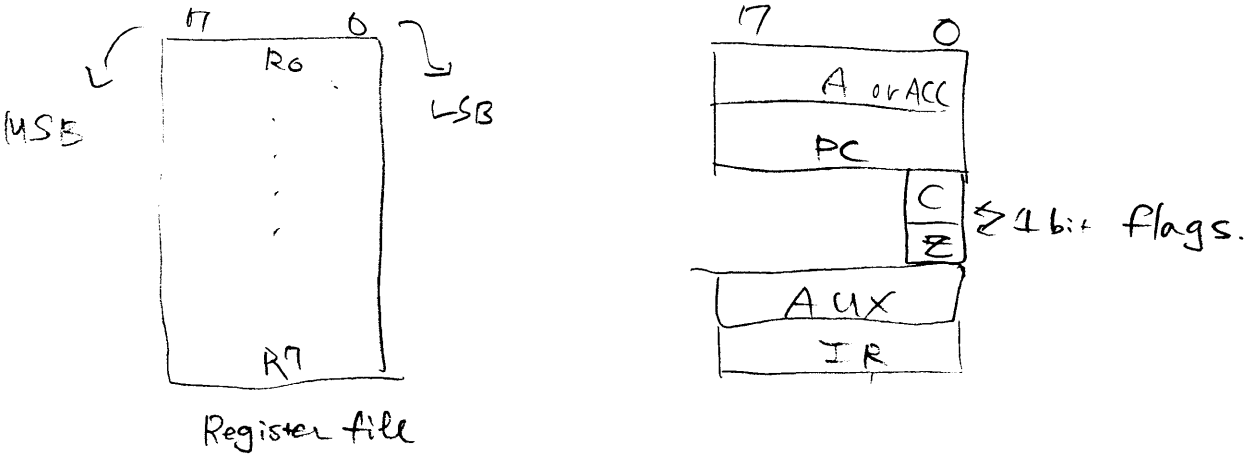
⭐ Slides & handouts can be downloaded from BB.

⭐ WIMP51 - Weekend Instructional Micro Processor ; an 8051 subset.

⭐ hand-out WIMP51 architecture)

The programmer sees WIMP51 as a collection of registers. ⇒ refer to as the programmer's model.

⭐ WIMP51 programmer's model



① 8-bit Accumulator (A or ACC)

② 8-bit registers R0 to R7

③ 8-bit program counter (PC)

④ Carry flag (C) - Used to store the carry out bit of an addition

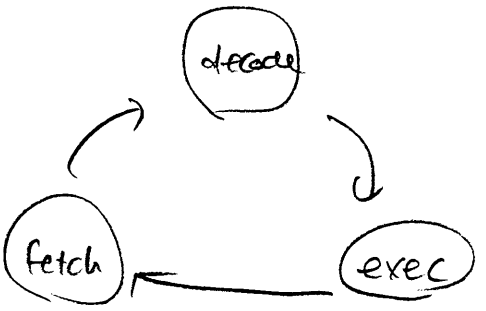
⑤ zero flag (Z) - If Acc is 0 then Z=1
If not Z=0
⇒ All Acc bits NOR'ed together.

⑥ 8-bit AUX register.

⑦ IR register : stores one instruction at a time.

✡ WIMP 51 Control Unit.

- Simple state machine with three states.
  - Fetch : fetch one inst from external inst mem
    Decode : decode the inst to determin what must be done
    Execute : execute the inst
  - For WIMP51, each state takes 1 clock cycle.
  - A comple loop is called an instruction cycle



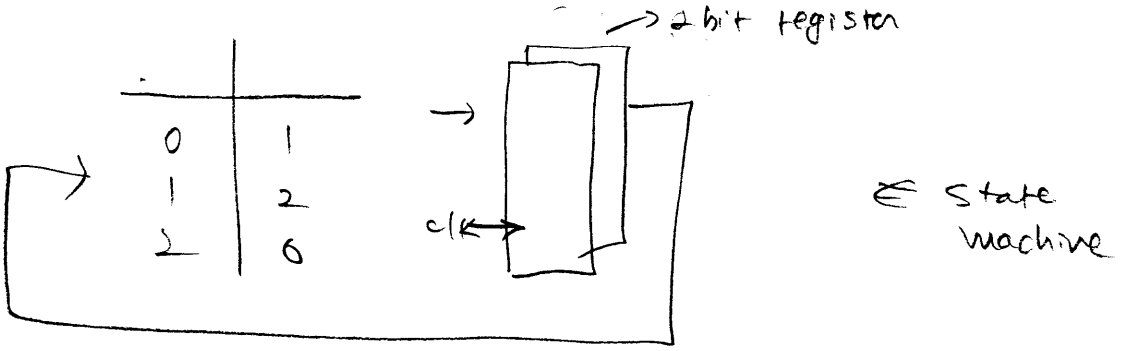
✡ Details on inst cycle.

Fetch cycle: the current instruction indicated by PC. is fetched from code memory and stored into the inst reg (IR).

Decode cycle: the inst is decoded and any required operands are fetched from memory or from the reg file. (called data fetch)

Exec cycle: the inst is executed. A result may be stored in a register or the ACC updated.

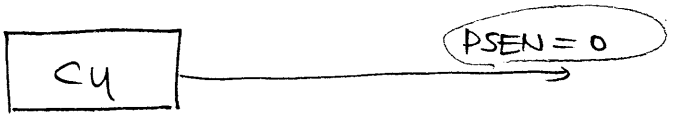
⭐ MIMPS1 Inst cycle Control unit (CU)



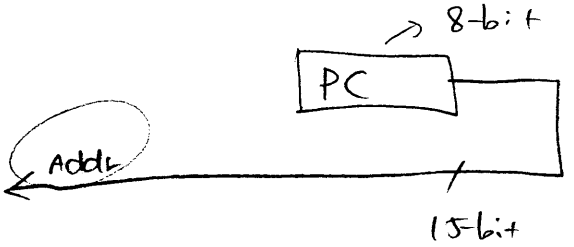
where 0 - Fetch, 1 - decode and 2 - execute.

⭐ Fetch cycle step 1 : addr ⇐ PC

- When the CU state machine is in Fetch cycle, PSEN = 0, when the active-low signal PSEN goes low (a 0), the addr bus (addr) is driven with the contents of the PC.



⭐ ⇒ we have enable signal 0 for the memory & addr to access the memory.



note that PC is 8-bit & addr bus is 15-bit.

⇒ If PC = 0000 1111 then addr = 000 0000 0000 1111

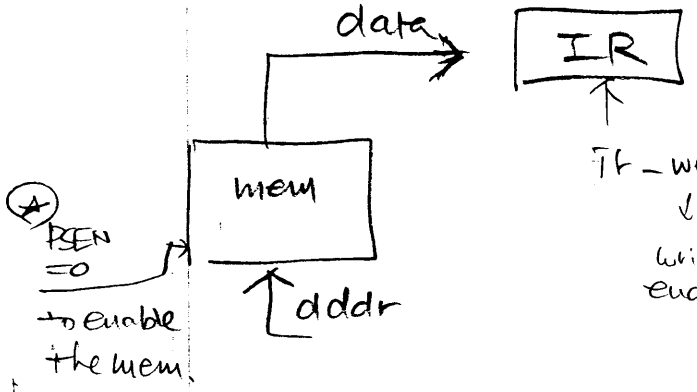
'0' is used to padd out.

⇒ since addr is 15-bit, 32KB memory can be used

④ Fetch cycle step 2 ① IR ∈ data

≈ IR ∈ C(Addr)

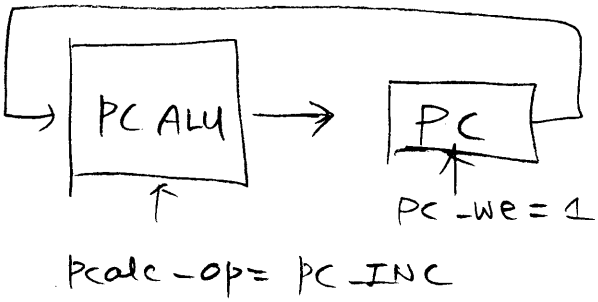
↓ contents of the addr location.



∈ this must be done to latch the register (positive edge triggered register).

② PC ∈ PC + 1.

④ while the next inst is being fetched the PC ALU control signal is set to PC_INC so that the PC will incremented at the end of the fetch cycle.



④ What if we want to "Jump"? AUX is used.

pc ∈ PC + AUX & Pcalc_op = PC_ADD

④ Decode cycle

The rising clock cycle which causes the CU (control unit) state machine to transition from the fetch to decode also latches the new inst into the IR.

Then inst decoder logic decode the inst.
(not shown in the diagram)

④ Inst set of WIMPS1

ASM code	Op code	8bit immediate data D	meaning
MOV A, #D	0111 0100 (74H)	d ··· d	A ∈ D
ADDC A, #D	00 11 0100 (34H)	d ··· d	C, A ∈ A+D+C
MOV Rn, A	1111 1nnn (F8~FFH)		Rn ∈ A
MOV A, Rn	1110 1nnn (E8~EFH)		A ∈ Rn
ADDC A, Rn	0011 1nnn (38~3FH)		C, A ∈ A+Rn+C
ORL A, Rn	0100 1nnn (48~4FH)		A ∈ A or Rn

load & add immediate insts (2nd byte must be fetched)

register destination inst.

register source inst

ANL A, Rn	0101 1nnn 58~5FH	A ∈ A and Rn	
XRL A, Rn	0110 1nnn 68~6FH	A ∈ A ⊕ Rn	

Single register no operand insts

SWAP A	1100 0100 C 4H	A ∈ A(3~0) & A(7~4) Swap two nibbles in A.
CLR C	1100 0011 C 3H	C ∈ 0
SETB C	1101 0011	C ∈ 1.

relative address insts (2nd byte must be fetched)

SJMP rel (rel: relative offset) — 1 0000000 (8-bit a…a) — PC ∈ PC + rel + 2 (to skip the current ins.)
8 0H

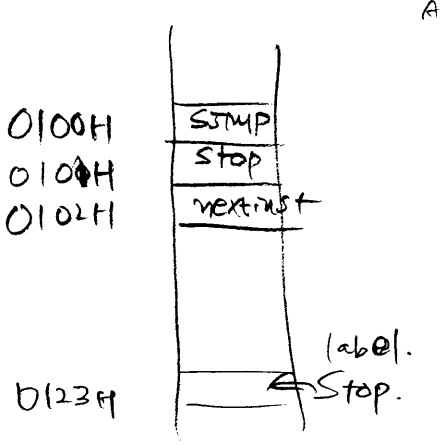
ex) The label "stop" is assigned to an inst at program mem location 0123H. The inst SJMP stop assembles into location 0100H.

After the inst is executed, the PC contains the value 0123H.

⇒ rel? (rel offset or rel addr)

0123H − 0100H − 2H

= (21H)



L JZ rel 0110 0000 a ... a pc ← PC + rel + 2
if (z = 1)
↓
ACC = 0.

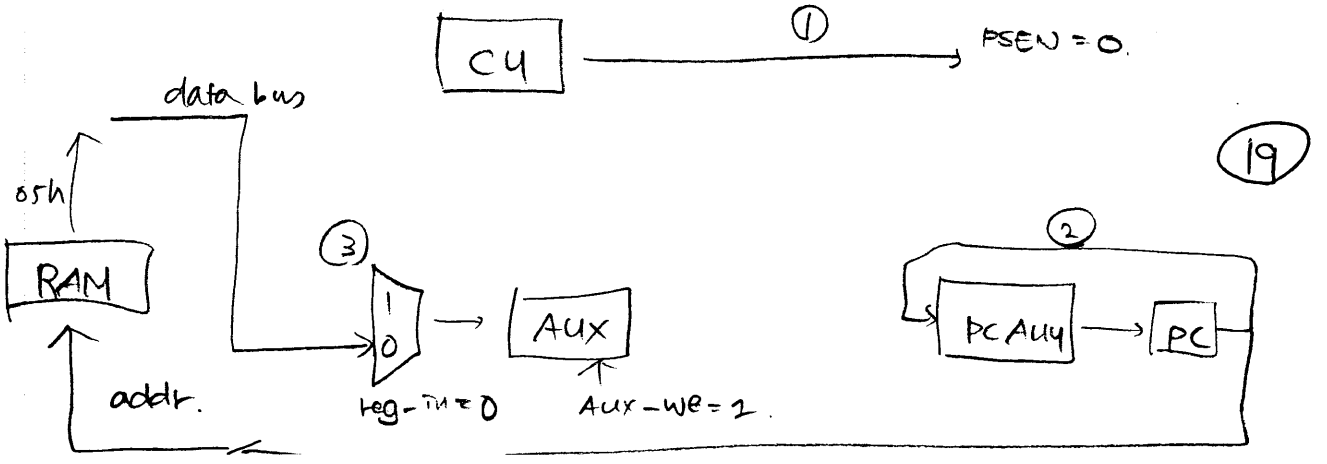
★ Decode cycle : Immediate Source. (: immediate mode insts)

ex) Mov A, #05h.
→ This inst moves data word 05h from the mem to Acc.

The inst (called opcode) mov is encoded as 0111 0100₂ or 74h.

① When the CU decodes this inst, it will drive PSEN low to cause a second byte to be fetched from memory. ② PC+1 is used to address the data.

③ The Aux mux control line is set to 0. to cause the AUX register to latch the data bus on the next rising clock edge. In this case the seond byte is 0000 0101₂ or 05h.



Decoder cycle : register source (register source ins's)

ex) MOV A, R5

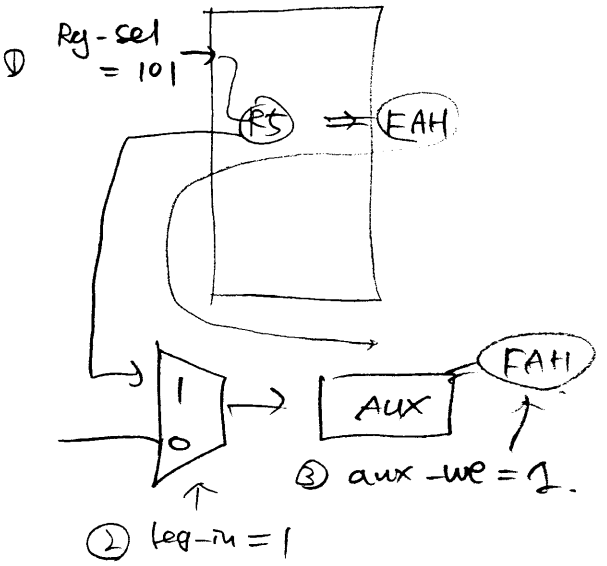
⇒ move contents of R5 to ACC.

IR contains 1110 1101 or EDh

① The lower three bits are used as reg-sel input to the reg file, to select R5 in this case.

② reg-in for AUX MUX is set to 1 to select reg file output.

When Aux-we = 1, the value of R5 will be latched into the AUX reg.



Execute cycle: register & immediat source insts.

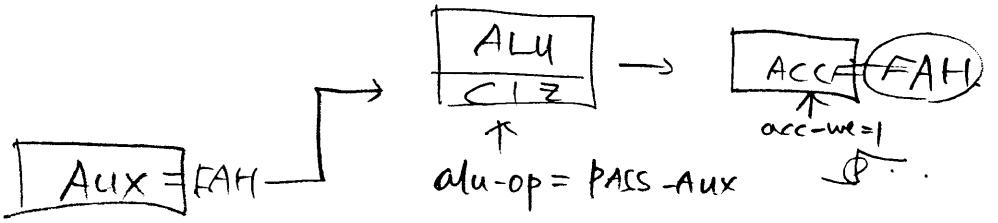
⇒ The AUX register contains some value from either mem or reg file

ex) MOV A, R5.

During the execute cycle, the operand which was fetched is now available on the AUX output bus. (FAH)

ALU is to be set up to pass the AUX output to the ACC. (alu-op = PASS_AUX must be used)

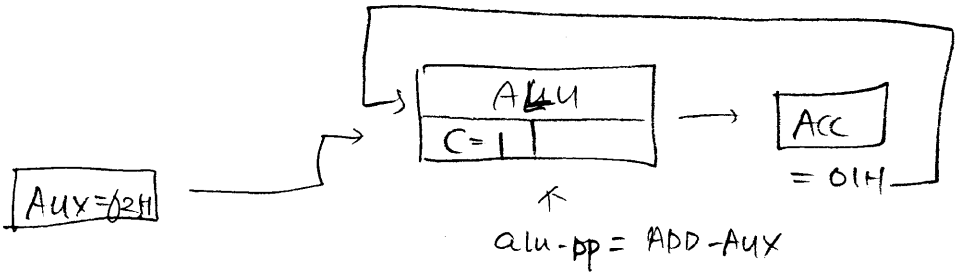
Next positive-edge (acc-we = 1) the value (FAH) will be latched in to ACC.



ex) ADDC A, R5.

suppose the current value of A is 01H
R5 is 02H
C is 1

alu-op = ADD_AUX must be used.



① ALU adds up AUX, C, ACC ⇒ 04H.

② Next positive edge latches the value into ACC = 04H.

Execution cycle for jump insts.

loop: SJMP loop

label for jump insts.

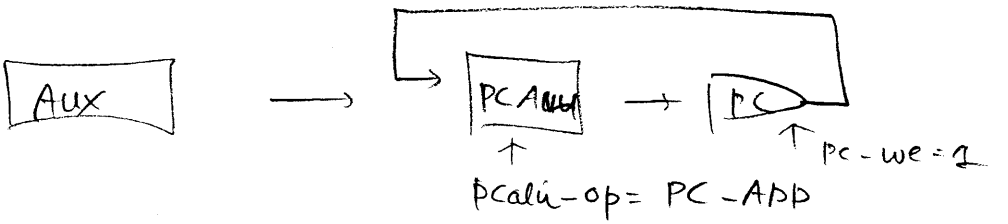
This inst jumps to inst labled by loop.

⇒ This inst jumps to itself.

For a jump inst, the value of the AUX register is added to the value of the pc.

Next rising clock edge, the pc will be updated.

Since jump insts are two bytes long, PC ← PC+2 must be done before the value of AUX is added.



for loop: SJMP loop ASM program, PC is initially 00H

then PC + 2 + AUX must be 00H. So, it can jump to itself. so, AUX value must be FEh. (since 02h + FEH = 00h and c=1)

↳ -2 in 1's complement #

⇒ So, loop: SJMP loop is encoded as.

80 FEH.

opcode rel addr

Ⓐ WIMP51 timing

① Regs get loaded by rising edge at end of cycle

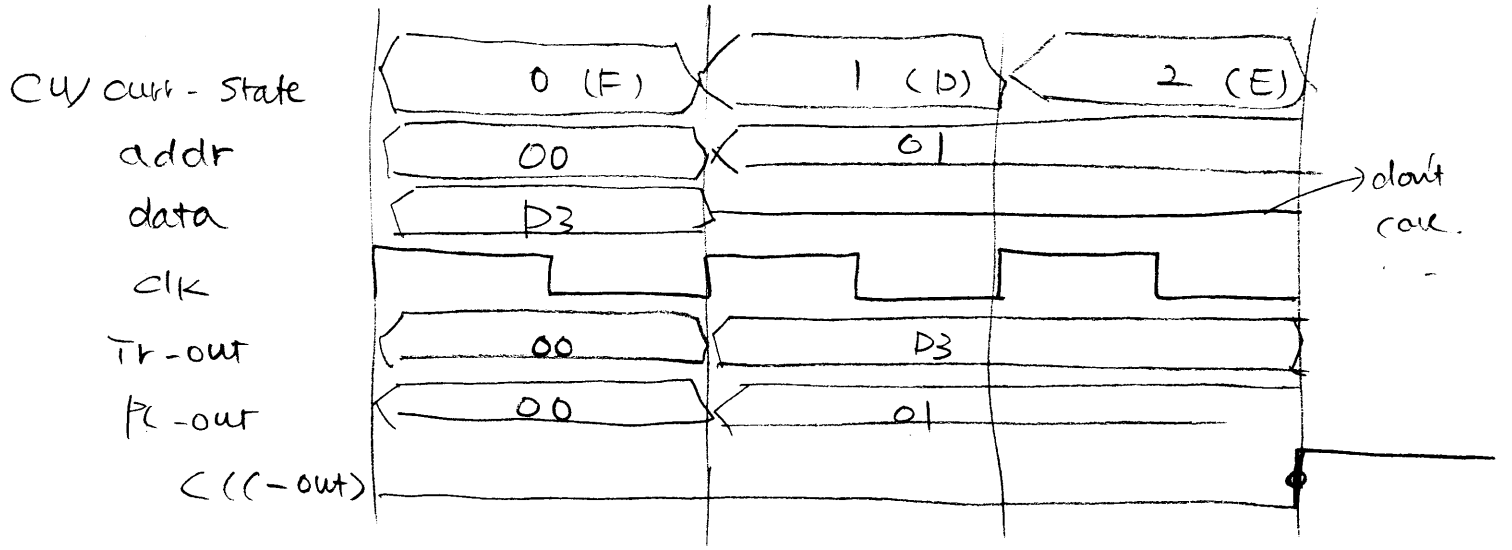
② Each inst lasts three clk cycles (F, D, E)

③ Each inst cycle consists of a F, D, E clock cycle.

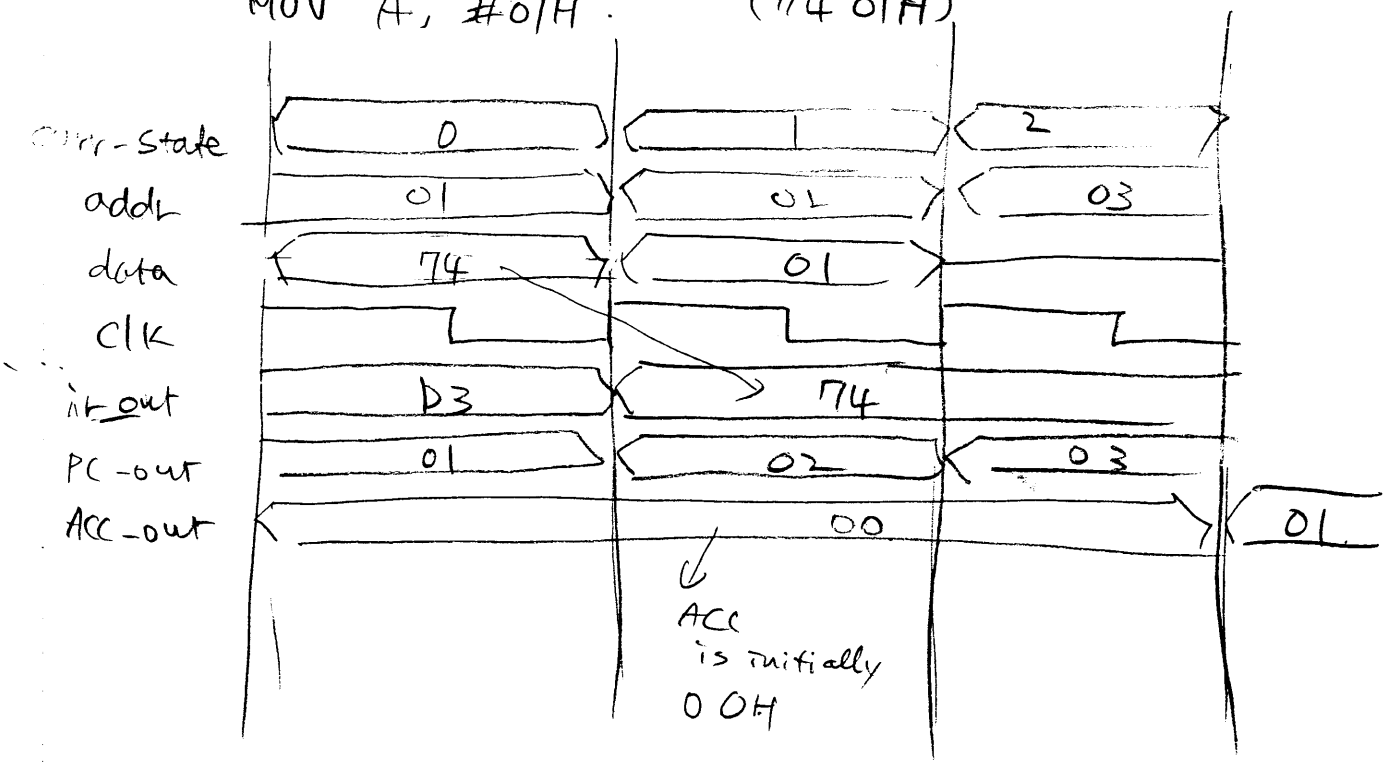
④ True 8051 is similar but more options so more complex (12 or more clock cycles per inst).

⇒ from the WIMPSI slides on p13.

ex) SETB C (D3h)



MOV A, #01H. (7401H)



★ Rapid prototyping concept for WIMPS1

- Timing diagrams generated by a simulator.
- Simulator executes a model of WIMPS1.
- Model is written in VHDL.
- VHDL can be used to synthesize WIMPS1 hardware
- Simulation models let us 'try out' a design before committing to HW
- we can simulate HW, SW, or both

★ WIMPS1 ← Assembly programming

ex) Calculate 2+1

addr	label	Instr	machine code		
0000h		MOV A, #01h	74	01	) load '1' to R0.
2h		MOV R0, A	F8		
3h		MOV A, #02h	74	02	– load '2' to ACC
5h		CLR C	C3		– clear C
6h		ADDC A, R0	38		– ADD R0 & ACC
7h		MOV R1, A	F9		– load '3' to R1
0008h	Stop:	SJMP Stop	80	FEh	– Stop.

★ "branch to self" halt WIMPS1 program since WIMS1 does not have 'halt' inst.

★ 08h + 02h + reladdr = 08h
So, rel addr = FEh. (-2_{10})
(of course, C becomes 1).

Summary of changes in Regs.

Facts	PC	R_0	R_1	Acc	C
MOV A, #01H	0	X	X	X	X
MOV R0, A	2	X	X	01H	X
MOV A, #2H	3	01H	X	01H	X
CLR C	5	01H	X	02H	X
ADDC A, R0	6	01H	X	02H	0
MOV R1, A	7	01H	X	03H	0
Stop: SJMP Stop	8	03H	03H	03H	0

ex) 3 X 5 with loop.

⇒ can be done by adding 5 to itself three times.

```
        MOV  A, #0      ← decimal immediate value
        MOV  R0, A      ← R0 is used to hold the product
        MOV  A, #3
        MOV  R1, A      ← R1 is loop counter
loop:   MOV  A, #5      ACC = 5
        CLR  C          C = 0
        ADDC A, R0      } R0 = R0 + 5
        MOV  R0, A
        MOV  A, R1      } counter = counter - 1
        CLR  C          } two's complement # . (-1_10)
        ADDC A, #0FFh   }
        MOV  R1, A
        JZ   stop       if counter = 0, go to stop
        SJMP loop       if not, loop.
Stop:   SJMP stop
```

Quiz #1 (2-3-03) (20 pts)

Consider the following WINBOND ASM program.

MOV A, #10	0111 0100 0000 1010	(7404h)
MOV R0, A	1111 1000	(F8h)
MOV A, #5	0111 0100 0000 0101	(7405h)
CLR C	1100 0011	(C3h)
ADDC A, R0	0011 1000	(38h)
MOV R1, A	1111 1001	(F9h)
MOV A, #6	0111 0100 0000 0110	(7406h)
ADDC A, R1	0011 1001	(39h)
STOP SJMP Stop	1000 0000 1111 1110	(80 FEh)

≈ -2 in 2's complement

① Write down the machine code for this program.

② List all registers that are used by this program and their contents for each inst (after execution).

PC	R0	R1	Acc	C
0	x	x	x	x
2	x	x	0Ah	x
3	0Ah	x	0Ah	x
5	0Ah	x	15h	x
6	0Ah	x	05h	0
7	0Ah	x	0Fh	0
8	0Ah	0Fh	0FA	0
10	0Ah	0Fh	06A	0
11	0Ah	0Fh	15A	0

HW #2.

Consider the WIMPS1 ASM program which calculates 3*5.

1) Write down the machine code for the pgm (30p)

2) How long does it take to execute the pgm (20p) in clock cycles?

3) list all registers that are used by this (70p) program and list their contents after each MST is executed.

⛤ The 8051 – Hardware Summary (ch2).

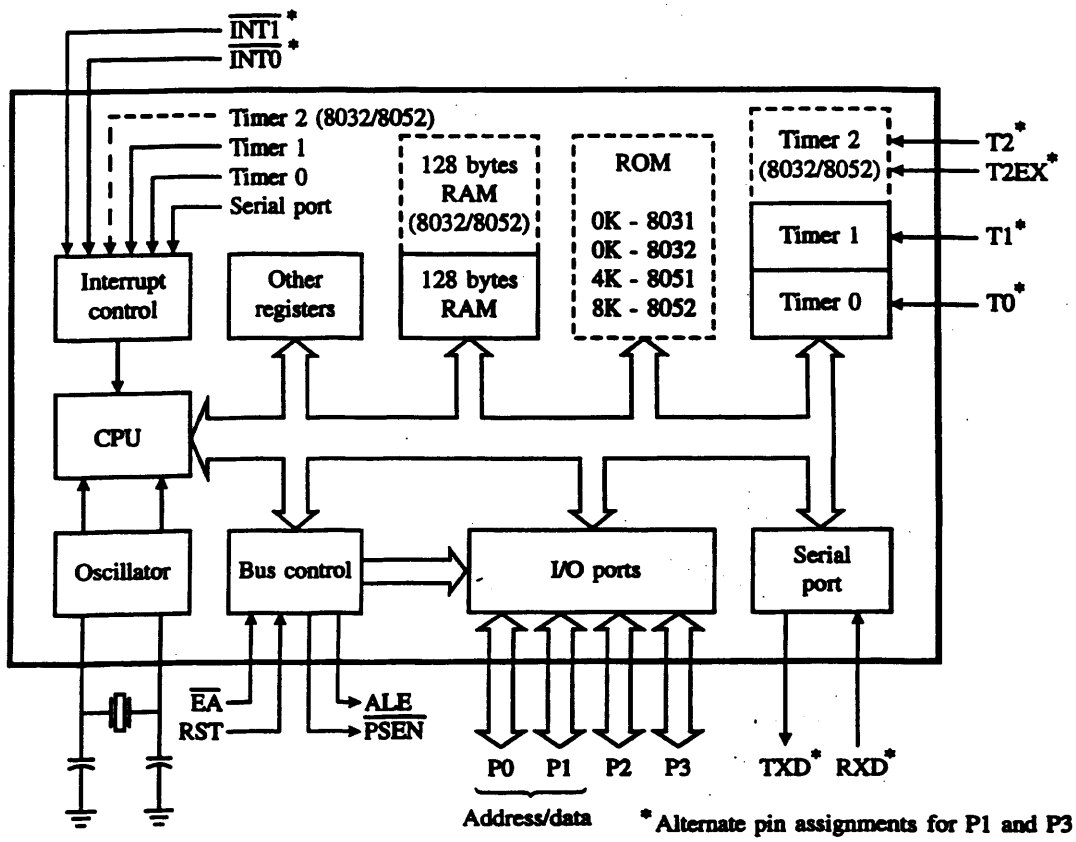
⛤ Handout 1 – 8051 Architecture.

- Diagram shows basic architecture of the 8051 MC.
- The CPU, central processing unit, is the processing core of the MC.
- In addition to CPU, most MCs also have internal osc., ROM, RAM, I/O ports, Serial ports, internal hardware timers.

⛤ 8051 family characteristics

	typical
✓ 0 ~ 32K internal ROM (code)	4K
✓ 4 ~ 1024 bytes internal RAM (data)	128 B
✓ 0 ~ 16M external code space	64K
✓ " data "	64K
✓ 1 ~ 6 8-bit I/O ports	4
✓ 0 ~ 8 16-bit timer/counters	2
✓ Serial communications interface	1 UART
⇒ Universal Asynchronous Receiver-Transmitter.	
✓ Boolean processor (operates on single bits)	ALL
✓ Internal bit-addressible memory	210 bits
✓ 4μs mult/div instruction	2
✓ 1 ~ 10 2-level ext interrupts	2
✓ 1 ~ 40 internal interrupts	2
✓ 0 ~ 40 MHz clock	4 MHz
✓ 1.8 ~ 5.5V operation	5V
✓ 6 ~ 12 clock machine cycle	12 clock

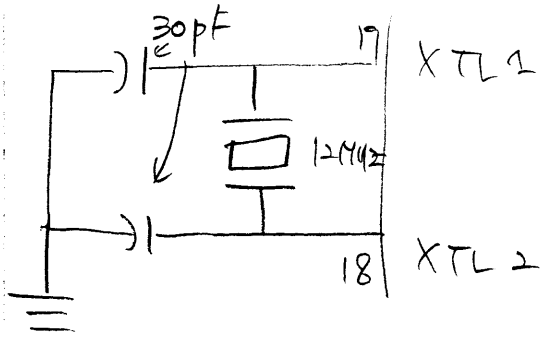
8051 Architecture



8051 pins

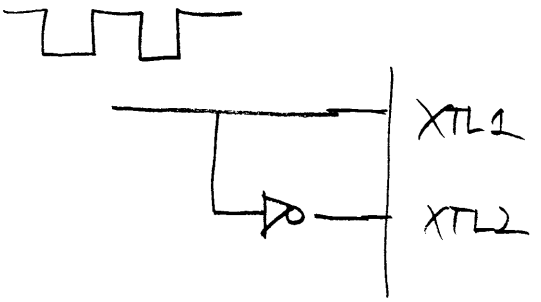
- clock XTL1 (19) & XTL2 (18)

① Method 1 – using on-chip oscillator.



2 Stabilizing capacitors + 1 crystal needed.

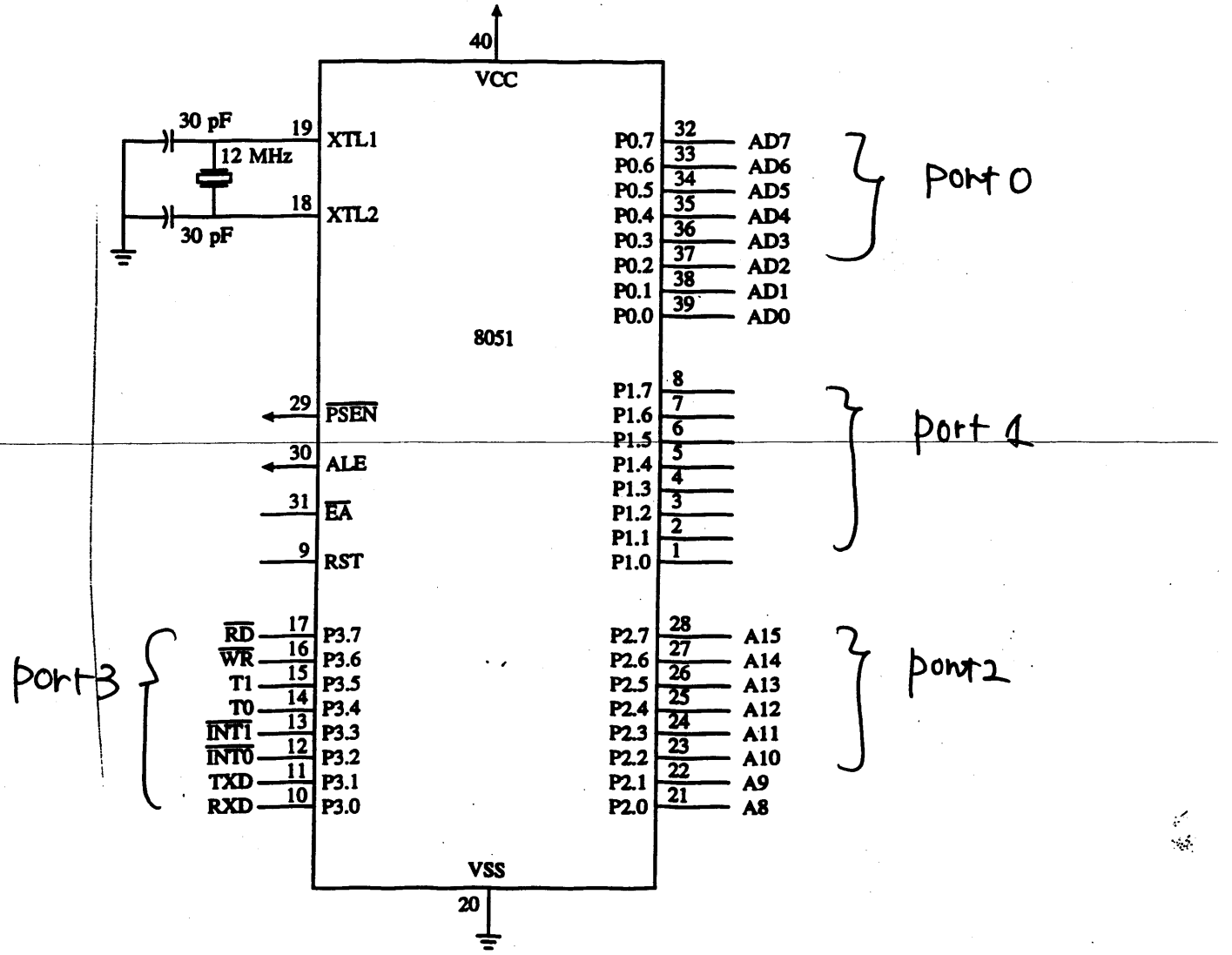
② Method 2 – using external clock source
(called TTL (Transistor – Transistor Logic)
Clock – 0V for logic 0 and 5V for logic 1)



- $\overline{PSEN}$ (29) (program store enable) –
goes low to fetch external code. (like WINPST)

- ALE (30) Address latch enable)
tells external device to latch
onto address. (will be disscussed later on).

PINOUT



Alternate pin functions for port pins

BIT	NAME	BIT ADDRESS	ALTERNATE FUNCTION
P3.0	RXD	B0H	Receive data for serial port
P3.1	TXD	B1H	Transmit data for serial port
P3.2	$\overline{INT0}$	B2H	External interrupt 0
P3.3	$\overline{INT1}$	B3H	External interrupt 1
P3.4	T0	B4H	Timer/counter 0 external input
P3.5	T1	B5H	Timer/counter 1 external input
P3.6	$\overline{WR}$	B6H	External data memory write strobe
P3.7	$\overline{RD}$	B7H	External data memory read strobe
P1.0	T2	90H	Timer/counter 2 external input
P1.1	T2EX	91H	Timer/counter 2 capture/reload

- $\overline{EA}$ (31) (External access)

  $\overline{EA}$ - 0 : access (only) external mem (code)

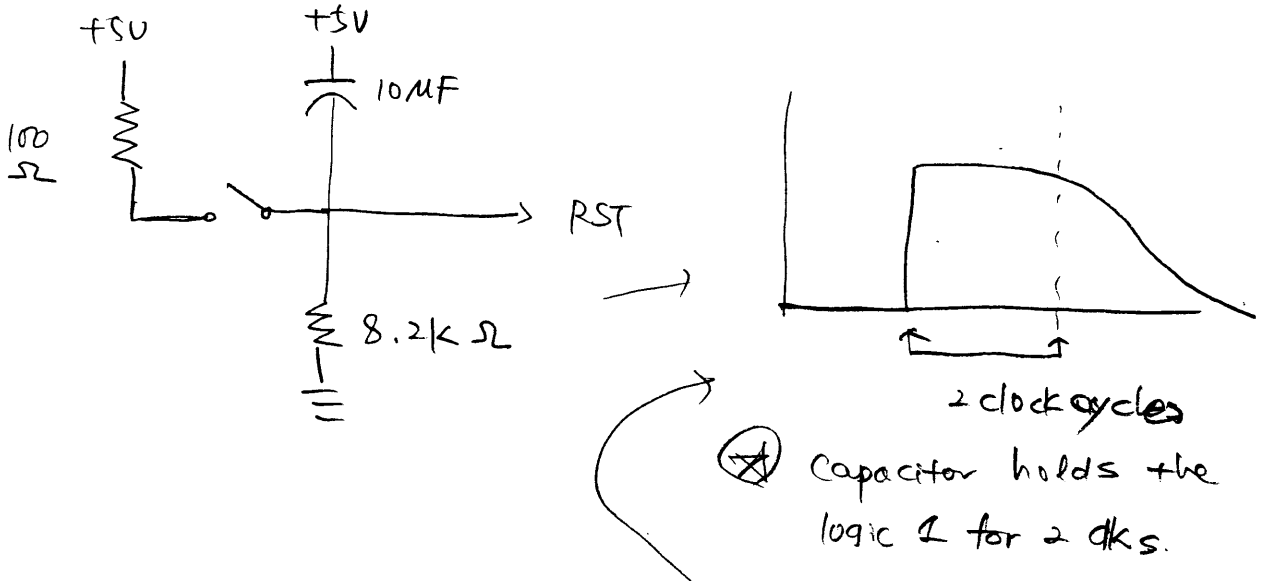
  $\overline{EA}$ - 1 : use internal mem, when possible (code)

- RST (9) (Reset)

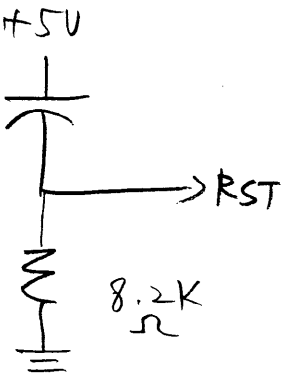
  RST = 1 for 2 clock cycles - reset

  RST = 0 - normal operation

① Method 1. - Manual reset



② Method 2. - power-on reset.



- Power connections.
  Vcc (40) +5V
  Vss (20) ground.

- Port 0 (32 ~ 39) General purpose I/O or Multiplexed address/data

- Port 2 (21~28) General purpose I/O or address (upper byte)

- Port 1 (1~8): G.P. I/O or sometimes timer 2.

- Port 3 (10~17): G.P. I/O or other functions (see 8051 handout).

(*) Port pin naming convention.
P0.1 — Pin 1 of port 0.

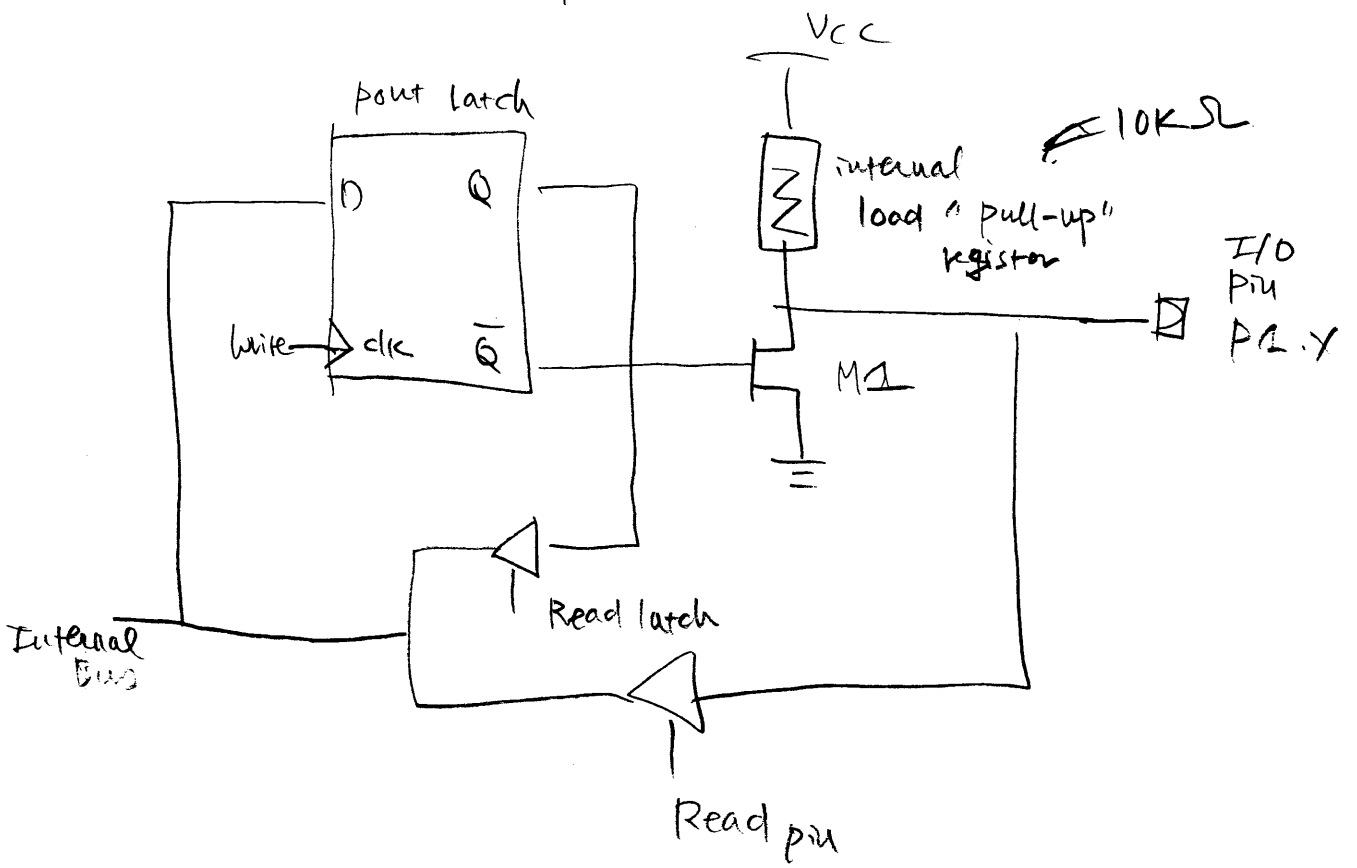
(*) I/O Port structure.

① 32 pins are alloted for 4 eight bit I/O ports.
— P0, 1, 2, 3.

② At power-on all are output ports by default.

③ To configure any port for input, write all 1's

(0XFF) to the port.



① For Input write all 1s to the port.

D = 1.

so, Q = 1

Q̄ = 0

M1 = OFF.

⇒ Read pin asserted by read inst.

② For output

write 1 to D → Q̄ = 0 → M1 = off

→ pullup "pulls" output weakly high (1)

← write 0 to D → Q̄ → 1 → M1 = on

→ tran pulls output strongly low (0)

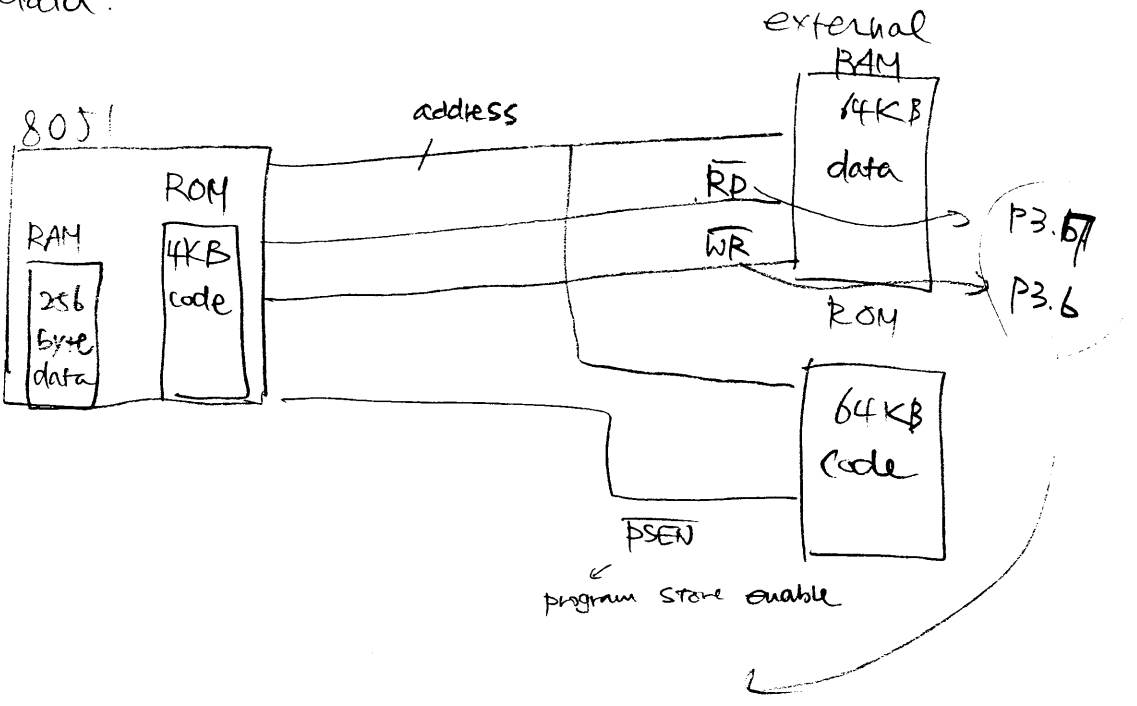
can damage the M1 if P1.X is Vcc ⇒ so, 10KΩ resistor is needed.

ex) 8051 Assembly program for I/O port.

```
MOV  A, #55H      move 55H to A
MOV  P0, A        write A to P0
MOV  P1, A
MOV  P1, #0AAH    immediate move is possible.
MOV  P0, #0FFH    configure P0 for input
MOV  A, P0        read from P0.
```

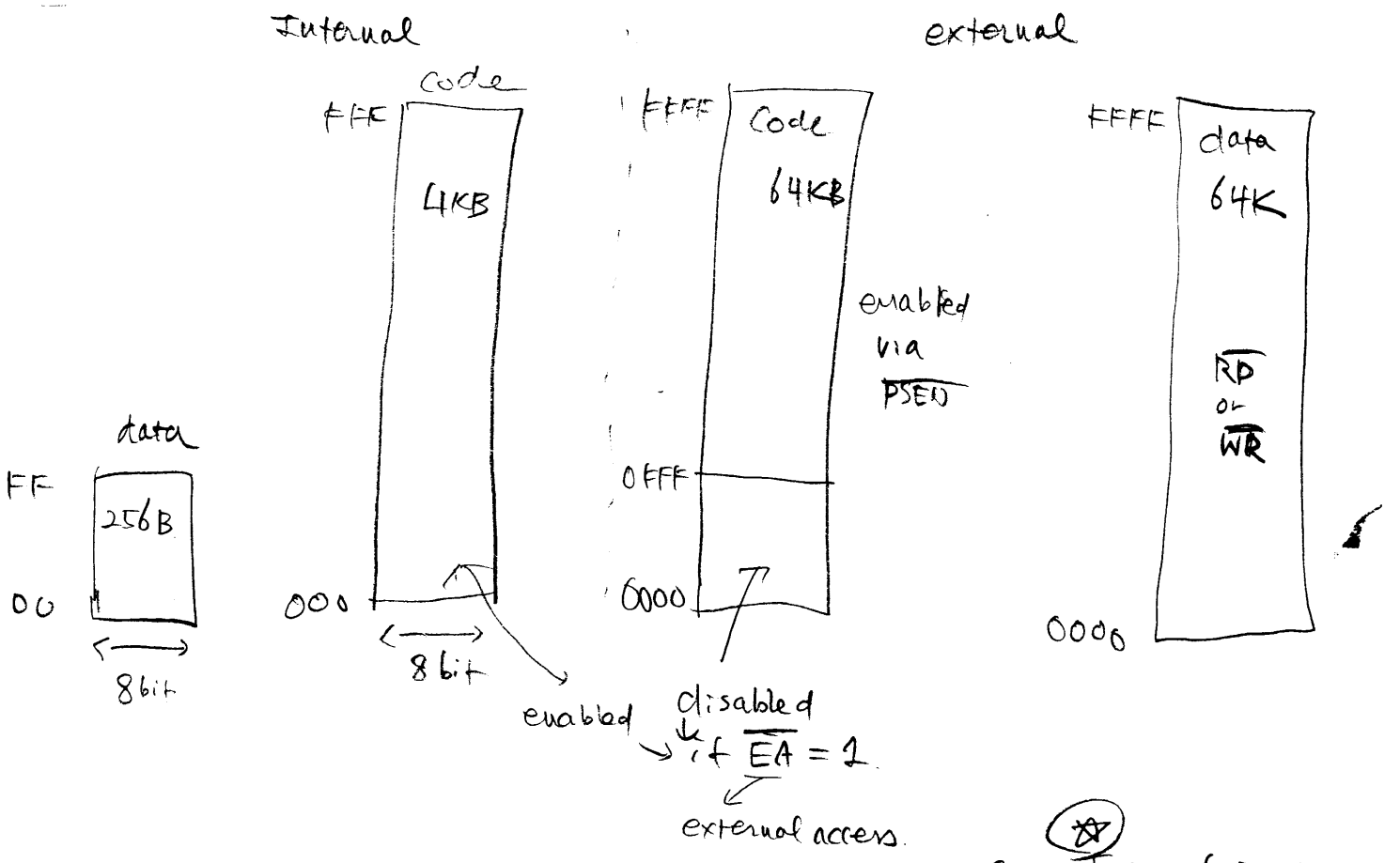
✡ Memory Organization.

8051 implements a seperate mem space for code & data.



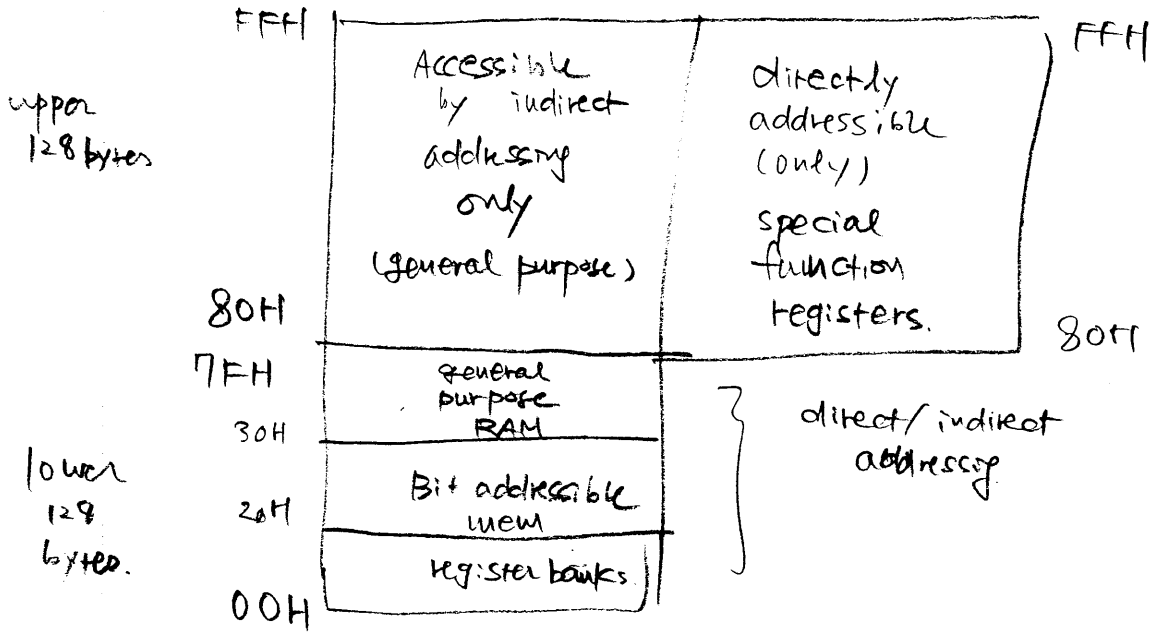
P3.6 : external data mem read strobe

P3.7 : " write "

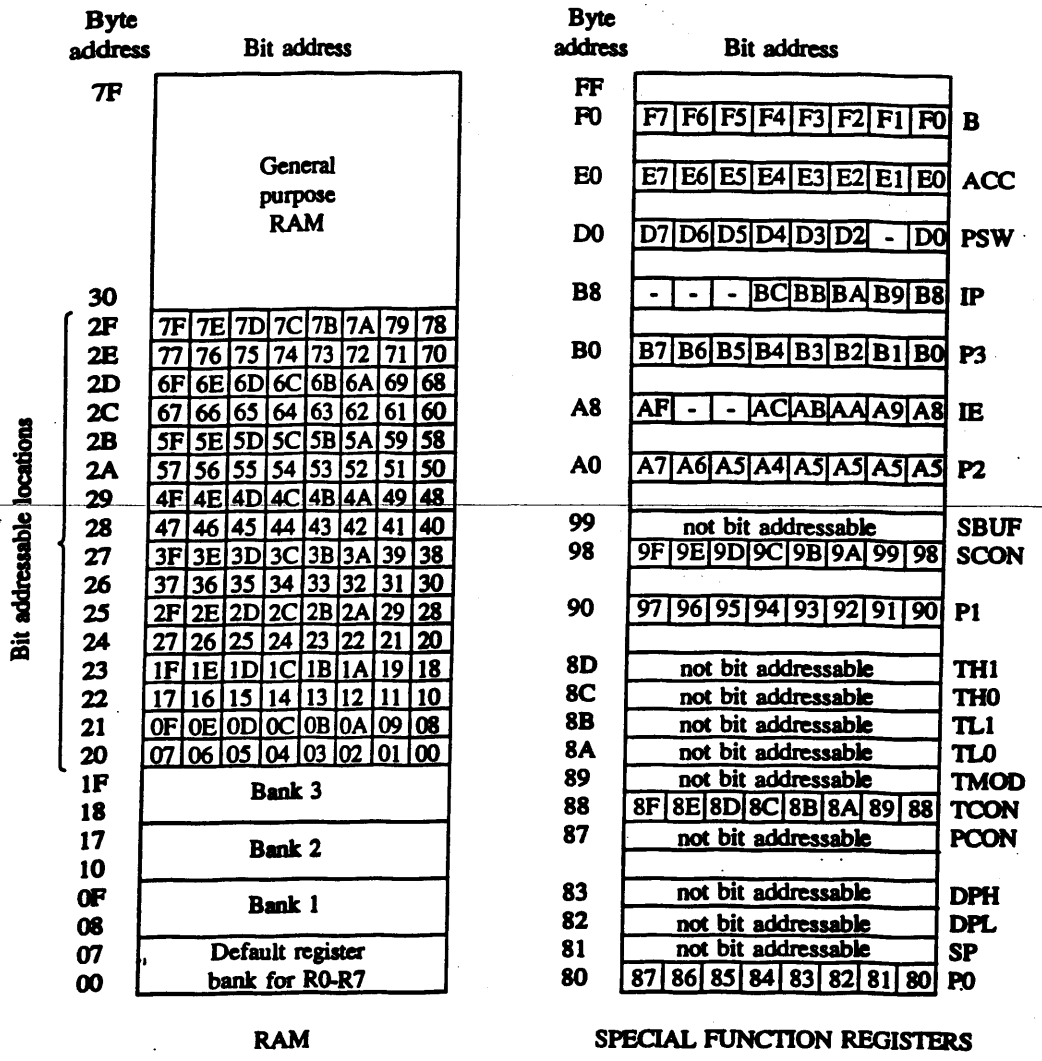


Internal RAM
hand out

✡ Internal RAM 00H ~ FFH



INTERNAL RAM



ex) direct addressing.

MOV A, 42H ← Move contents of mem (42H) into A.

A ← MEM(42H) ← mem contents of

ex) indirect addressing

MOV R0, #42H

MOV A, @R0

R0 ← #42H

move contents of location pointed by R0 into A

A ← Mem (mem (R0))

⊛ Bit addressable RAM space. 20 ~ 2F

Individual bit has bit address 00H ~ 7FH.

ex) Setb 67H

⇒ mem bit at 67H becomes 1.

ex) "bit address 67H" is the MSB at "byte address 2CH"

so, to set bit 67H.

bit pattern

```
MOV   A, 2CH
ORL   A, #1000 0000B   ← 1
MOV   2CH, A
```

ex) what inst would be used to set bit 3 in byte address 25H?

Set B 2BH

Note that the LSB is bit 0.

✡ Register banks.

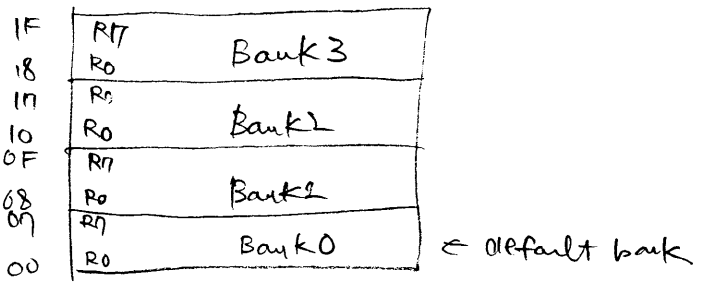
8051 has 8 regs : R0 ~ R7

By default 00H ~ 07H are used.

To allow context switching, 8051 has 3 more banks

↙

used to quickly switch between subroutines.



ex)

default bank's R0.

MOV A, R0

specific bank can be selected by.

bank 0	bank 1	bank 2	bank 3
MOV A, 00H	MOV A, 08H	MOV A, 10H	MOV A, 18h.

ex) what is the addr of regs in reg bank 3?

⇒ 1 Ch.

indirect : General purpose Reg.

→ direct : SFR.

✡ Special function registers (80h FFh).

See internal RAM handout.

; not just a mem location - a reg serving a specific function

They don't take up the entire memory space.

SFRs ending w/ 8 or 0 are bit addressible

⊛ (DPH + DPL) = DPTR : data pointer (16 bit)
DPH ↓ 83H, DPL ↓ 82H

: used to access external code or data mem.

ex) Write 55H into ext. RAM location 1000H.

```
MOV   A, #55H          load immediate data 55H to A.
MOV,  DPTR, #1000H     load 1000H to DPTR
MOVX  @DPTR, A         Move contents of A to mem (DPTR).
```

MOVX ↙ Move inst for external data mem

⊛ PSW - Program status word. (D0H)

D7H							D0H
CY	AC	F0	RS1	RS0	OV	—	P

- CY : Carry flag : set when carry out from bit 7 of ACC.

ex)

	ACC	C
	1111 1111	0
ADD A, #1	+1	
	(1) 0000 0000	→ (1).

→ Binary coded decimal

- AC - Auxiliary carry set for BCD values
  set if ① a carry was generated out of bit 3 to bit 4
  or ② if lower nibble is AH ~ FH.
  ↙
  Invalid BCD value.

ex) MOV R5, #2
MOV A, #9
ADD A, R5
⇒ AC? Acc?

R5: 0000 0010
A: 0000 1001
A. 0000 1011

Acc = 0AH & AC = 1. since AH is not a valid BCD value

DA A (decimal adjust acc) brings results grater than 9 back into range.

ex) A: 0000 1010.

DA A makes A ...

0001 0000 = 10H
Which is 10_{10} in BCD format.

- F0 ~ general purpose flag

- RS1, RS0 - Register bank select

RS1	RS0	Bank
0	0	0
0	1	1
1	0	2
1	1	3

ex) The following insts enable reg bank 3 and then move the content of R7 (Byte addr 1FH) to ACC.

```
SETB  RS1
SETB  RS0
MOV   A, R7
```

ex) to select Reg bank 2....

```
SETB  RS1
CLR   RS0.
```

- OV - overflow : set after an addition or subtraction operation if there was an overflow (out of range of +127 ~ -128).
  ⇒ note that MSB is 0 ⇒ positive
  " 1 = negative.

ex)
$$\begin{array}{rll} 0F & = & 15_{10} \\ +\ 7F & = & 127_{10} \\ \hline 8E & = & -116_{10} \end{array}$$

→ 2's complement #

⇒ OV bit is set

ex)
```
MOV R7, #0FFH
MOV A , #0FH
ADD A , R7
```

$$\begin{array}{rll} FFH & = & -1_{10} \\ +\ 0FH & = & 15_{10} \\ \hline (1)\ 0EH & = & 14_{10}. \end{array}$$ ← valid result.

⇒ OV = 0, ACC = 0EH, C = 1

- P ~ parity bit : Set to:
  1 ~ odd # of '1's in Acc
  0 ~ even # "

⇒ So, the # of 1's in the Acc + P is always even!

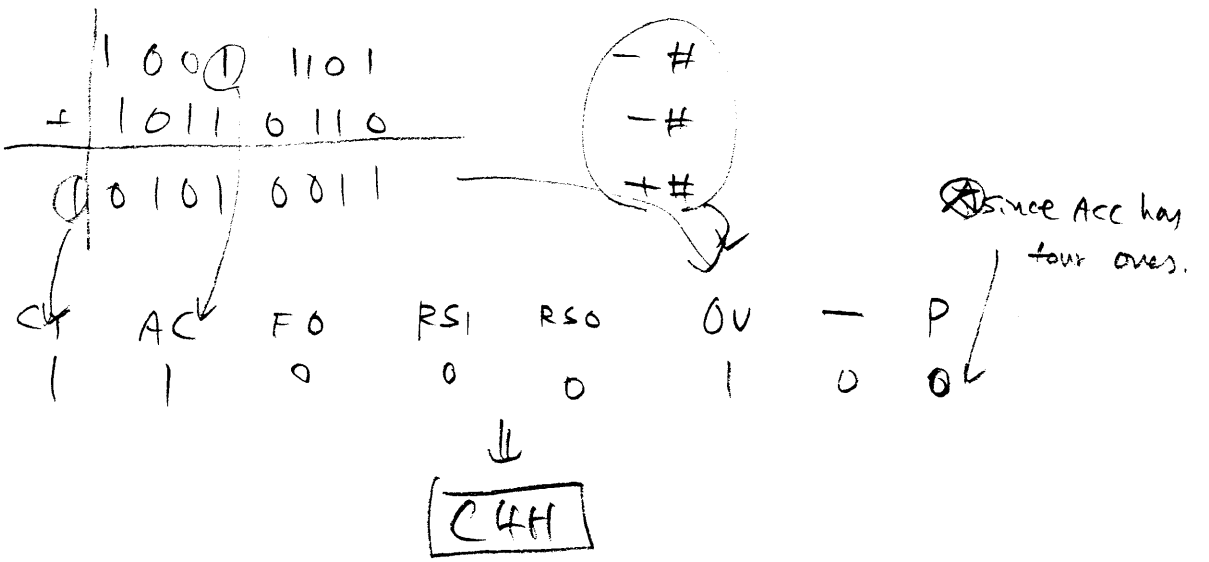
ex) A has 10101101B
then P is set to 1.

⇒ most commonly used to in conjunction with serial port

routines for error checking.

Ex) PSW = 0000 0000B
what is PSW after executing

```
MOV A, #9DH
ADD A, #B6H
```



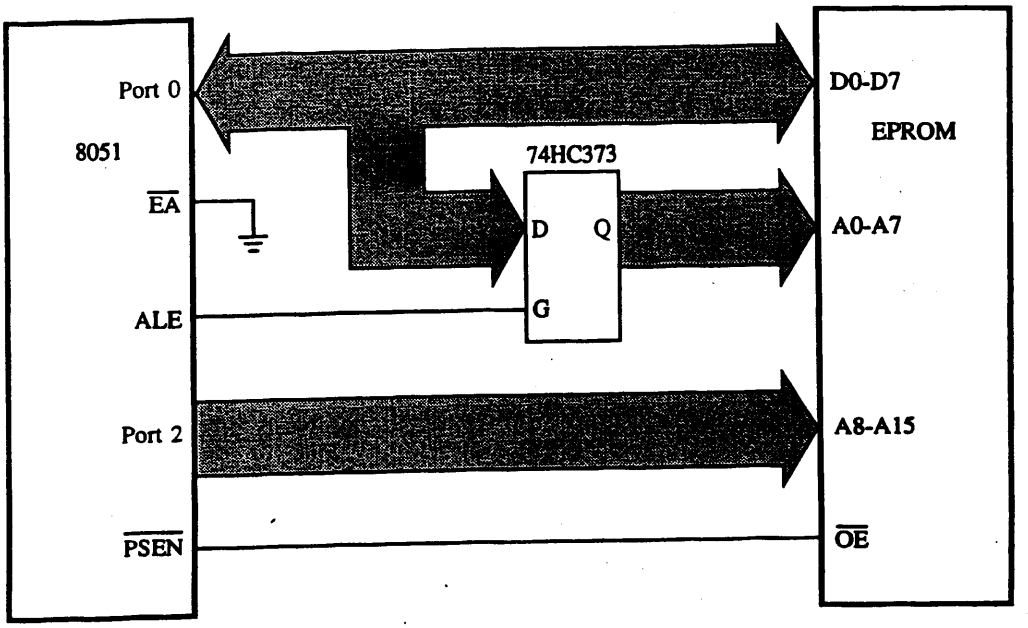


FIGURE 2–8
Accessing external code memory

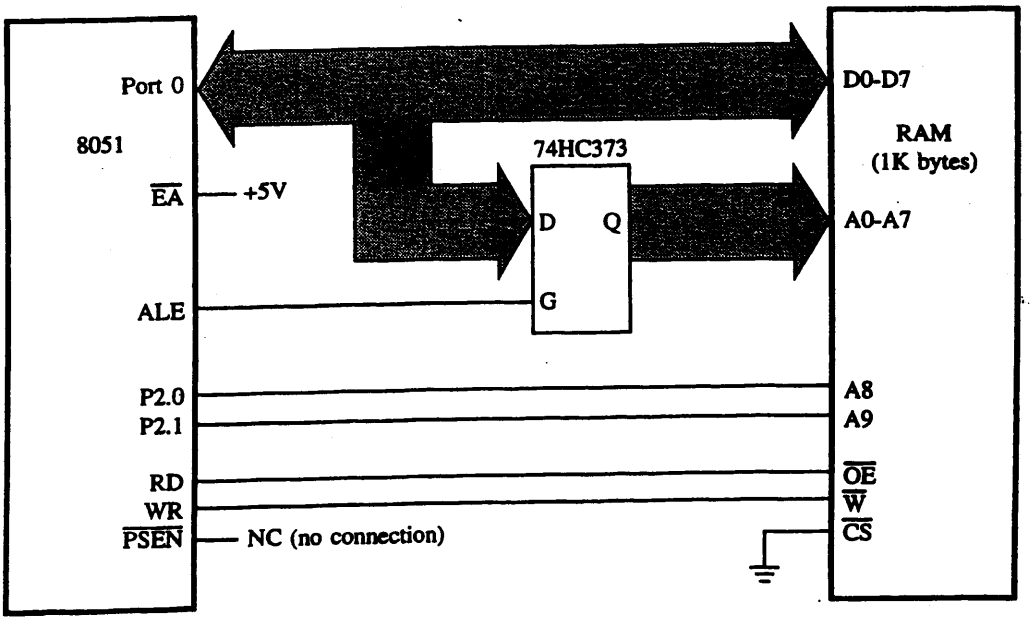


FIGURE 2–11
Interface to 1K RAM

(A) See handout Fig 2-8 (textbook)

(B) Accessing ext code mem.

- port2 is used as AH (address high 8 bits)
- port0 is used as AL (" low ")
  and data.

If port0 is used as AL, ALE (addr latch enable) must be high.
If ALE ↓, level-triggered latch 74HC 373 latches lower 8-bit addr and forward it to A0 ~ A7.

0) ALE, $\overline{PSEN}$ both high to find the addr location

1) write address

P0 = PCL = AL
P2 = PCH = AH

PC (16-bit)

PCH	PCL
8 bit	8 bit

P2 → Ports
PCH → Program Counter contents
AH → address bus Contents

2) latch AL externally when ALE ↓.
(74HC373 is level triggered).

3) P0 ready for input if reading

4) $\overline{PSEN}$ → low, then data read at 8051.

→ repeat for 2nd byte. (most insts are 2 bytes
8051 always fetch 2 bytes at a time)

⇒ If 1 byte inst, fetch 2 more dummy byte.

⊛ Accessing ext RAM (data mem)

0) $\overline{RD}$, $\overline{WR}$ high

1) P0 = DPL = AL
   P2 = DPH = AH
   } DPTR → data pointer

2) latch AL externally when ALE ↓

3) P0 ready for input if reading
   if writing : P0 = data written

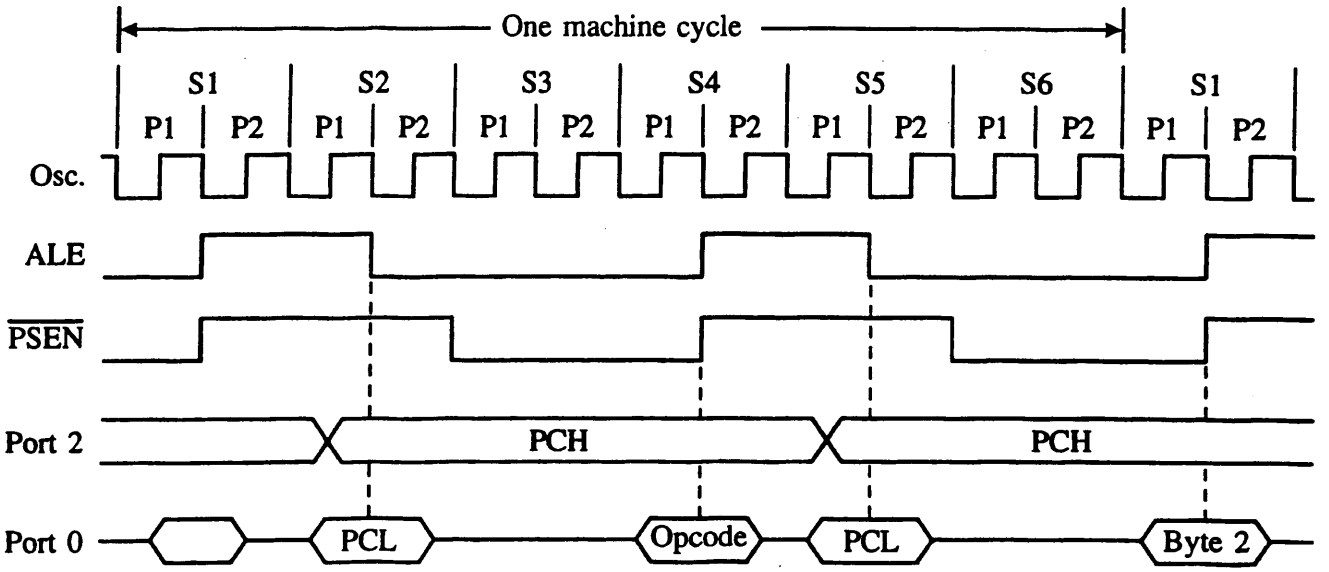
4) $\overline{RD}$ = low if reading
   $\overline{WR}$ = low if writing

⊛ Read timing for ext code mem.

For 8051...

1 machine cycle = 6 states
1 state = 2 clock cycles
⇒ 1 machine cycles = 12 clock cycles
= 12MHz is 1μS.

1 inst (instruction cycle) takes 1 ~ 4 machine cycles ≈ 12 - 48 clock cycles.



Note: PCH = Program counter high byte
PCL = Program counter low byte

FIGURE 2–9
Read timing for external code memory

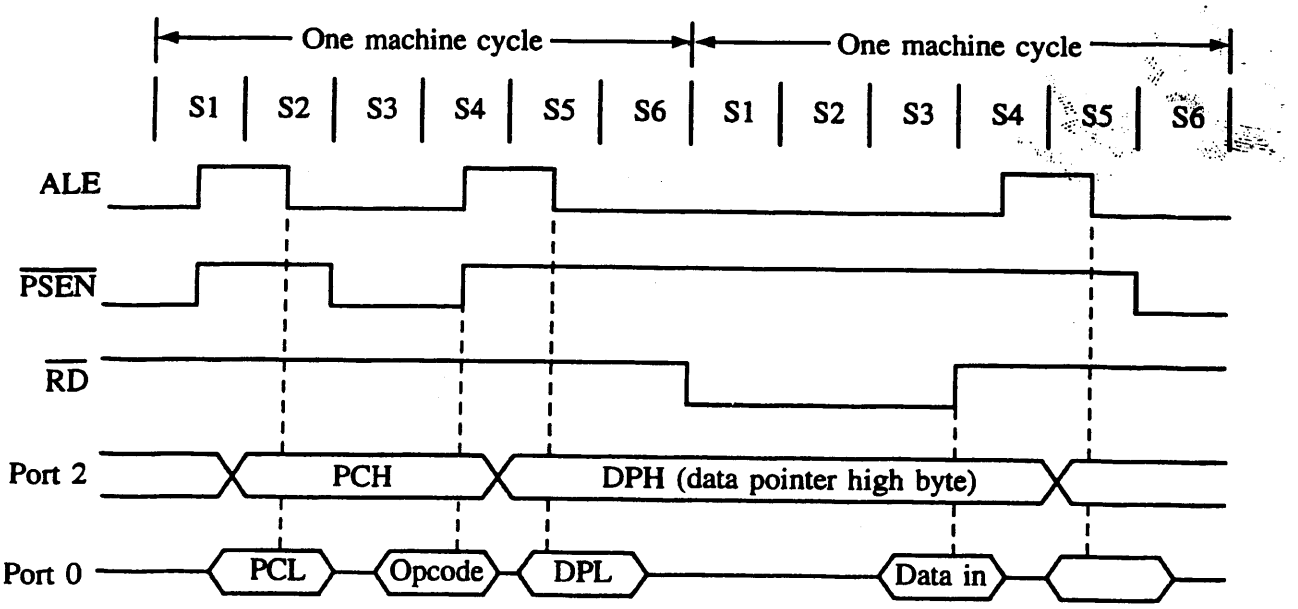


FIGURE 2–10
Timing for MOVX instruction