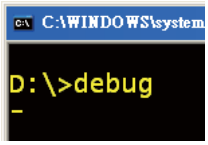
開始→執行→cmd



AX, BX, CX, DX = registers 寄存器
AX=AH+AL, …, DX=DH+DL

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B42 ES=0B42 SS=0B42 CS=0B42 IP=0100  NV UP EI PL NZ NA PO NC
```

Instruction Pointer
下一執行句

```
-u
0B42:0100 B401      MOV     AH,01
0B42:0102 CD21      INT     21
0B42:0104 CD20      INT     20
0B42:0106 CC        INT     3
0B42:0107 F6C702    TEST    BH,02
0B42:010A 7548      JNZ     0154
```

低階語言 Low Level Languages

記憶體地址 address　機器碼 machine code　匯編語言 Assembly Language

```
// getche()
// 輸入字符
mov ah,01
int 21
int 20
```


D:\>DEBUG

AX, BX, CX, DX = registers 寄存器

```
-u      -a
        0C8A:0100 mov ax,0200
        0C8A:0103 mov dx,0041
        0C8A:0106 int 21
        0C8A:0108 int 20
        0C8A:010A

        -h 010A 0100
        020A  000A

        -n test.com

-r      -r cx
        CX 0000
        :000A

        -w
        Writing 0000A bytes
        -q
```

AX=AH+AL

41(hex)
65(dec)
DX='A'

INT 21, AH=02
putc(DX); 輸出

結束
return(0);

計算
010A+0100 = 020A
010A−0100 = A(10)

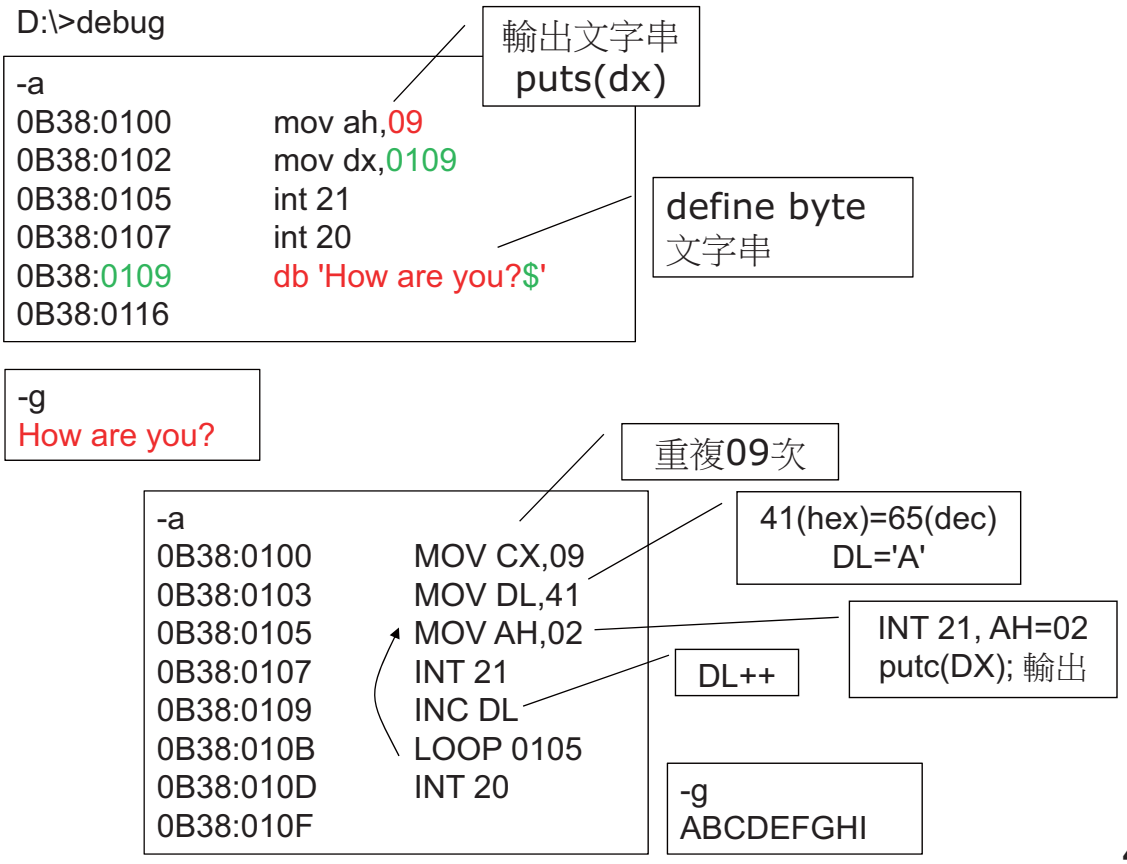
檔名

更改
CX=10

存檔

quit

```
D:\>test.com
A
```



-e 100 輸入資料 enter hex or char data
-e cs:100 "This is a string."

-u 列出指令
translates memory into mnemonics.

-d 100 顯示 dump memory
-g 執行 go/execute

Status Register 狀態寄存器

Set (1)	***Clear (0)***
OV = Overflow	NV = No Overflow
DN = Direction Down	UP = Direction Up
EI = Interrupts Enabled	DI = Interrupts Disabled
NG = Sign Flag negative(-)	PL = Sign Flag positive(+)
ZR = Zero	NZ = Not Zero
AC = Auxiliary Carry	NA = No Auxiliary Carry
PO = Odd Parity	PE = Even Parity
CY = Carry	NC = No Carry

? (Help)
A (Assemble)
C (Compare)
D (Dump)
E (Enter)
F (Fill)
G (Go)
H (Hex arith)
I (Input)
L (Load)
M (Move)
N (Name)
P (Ptrace)
Q (Quit)
R (Register)
S (Search)
T (Trace)
U (Unassemble)
W (Write)

開始→執行→cmd

```
D:\>debug
-
```

文字輸出

```
0100 MOV AH,09
0102 MOV DX,0113      ; puts(message)
0105 INT 21
```

輸入字符

```
0107 MOV AH,07        ; AL=getch()
0109 INT 21
```

AL=1B

```
010B CMP AL,1B        ; AL==ESC(1B=27)?
010D JZ  0111         ; 相同 jump zero 0111
010F JMP 0107         ; 不同 jump 0107
0111 INT 20           ; stop

0113 DB "press a key, esc to quit $"
```

JNZ 0107

```
printf("press a key, esc to quit ");
do{
        c = getch();
}while(c!=ESC);
```


C:\>debug

int 10

```
0100 MOV AH,02--| set cursor position
0103 MOV DH,06--| row(y) 6,
0105 MOV DL,40--| col(x) 40(=64)
0107 MOV BH,00--| page no.
0109 INT 10
```

```
// clear
// screen
mov ax,600
mov cx,0
mov dx,184f
mov bh,07
int 10
int 20
```

```
0100 MOV AH,09--| print char
0102 MOV AL,41--| letter A(41)
0104 MOV BH,00--| page no.
0106 MOV BL,FC--| attribute color
0108 MOV CX,05--| display 5 times
010B INT 10
010D INT 20
```

http://www.ablmcc.edu.hk/~scy/home/javascript/text-color.htm

```
-a 100
xxxx:0100 jmp 126          ; Jump to 0126
xxxx:0102 db 0d,0a,"This is my first DEBUG program"
xxxx:0123 db 0d,0a,"$"
xxxx:0126 mov ah,9         ; puts 輸出文字
xxxx:0128 mov dx,102       ; address of data(string)
xxxx:012B int 21           ; execute 執行
xxxx:012D mov ah,0
xxxx:012F int 21           ; Terminate 結束 Program.
xxxx:0131
-g=100
```

char s[50]="\nThis is my ... \n";

```
_
This is my first DEBUG program!
_
```


int 21 (debug)

http://www.ctyme.com/intr/int.htm

AH=01 AL=getche() 輸入	AH=02 DL=char 輸出 putc (DL)
AH=07 AL=getch() 輸入	AH=09 DX=addr 輸出 puts (DX) db "abc" define byte

http://www.youtube.com/watch?v=ijno5kDk1Xc

```
-u 100
0BA5:0100 B401      MOV     AH,01
0BA5:0102 CD21      INT     21
0BA5:0104 B402      MOV     AH,02
0BA5:0106 88C2      MOV     DL,AL
0BA5:0108 CD21      INT     21
0BA5:010A 80FA30    CMP     DL,30
0BA5:010D 7402      JZ      0111
0BA5:010F EBEF      JMP     0100
0BA5:0111 B44C      MOV     AH,4C
0BA5:0113 CD21      INT     21
```

getche()

putc()

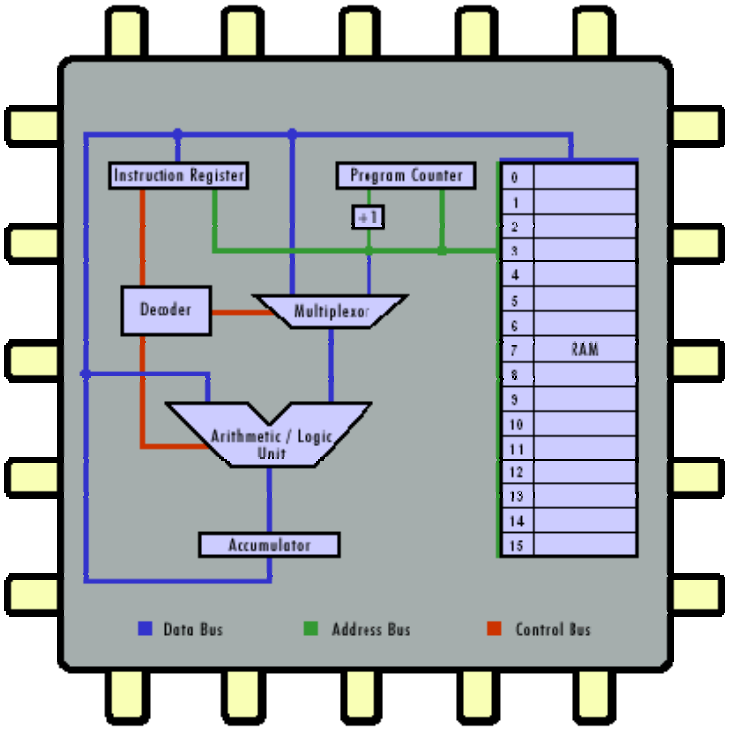
DL=='0'?

JMP=jump

JZ=
jump zero

記憶體地址 address　機器碼 machine code　匯編語言 Assembly Language

指令週期 Instruction cycle 機器週期 Machine cycle:



CPU

高階語言：C

```
for (n=0;n<=5;n++)...
```

低階語言：匯編語言　　寄存器 ACC　0

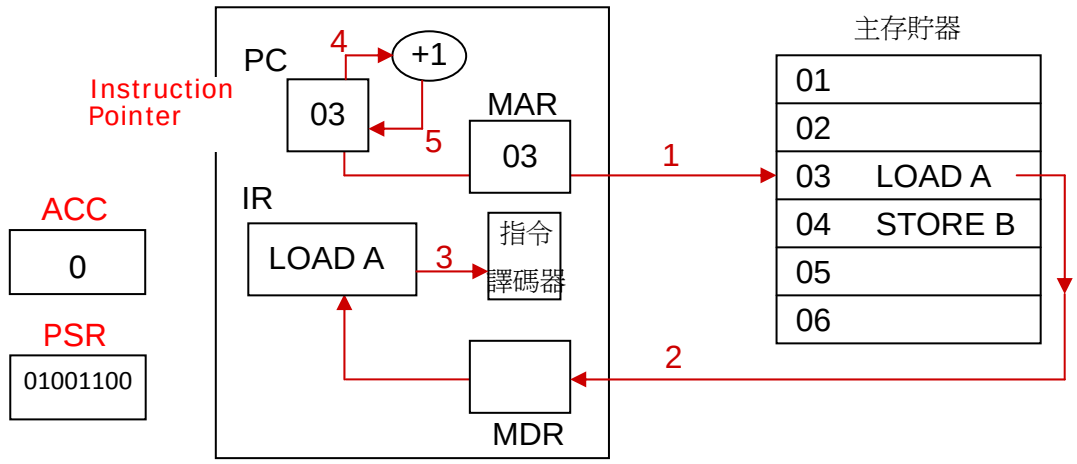
	memory[]	
0001	LOAD #5	// ACC=5
0002	STORE 0020	// memory[20]=ACC // MAX=5
0003	LOAD #0	// ACC=0
0004	STORE 0030	// memory[30]=ACC // N=ACC
0005	SUB 0020	// ACC=ACC-memory[20]
0006	JPZ 0010	// if(ACC==0) jump to 0010
0007	LOAD 0030	// ACC=memory[30] // N
0008	ADD #1	// ACC=ACC+1
0009	JMP 0004	
0010	STOP	
...		
0020	5	// MAX
...		
0030	0	// N

LOAD 載入　STORE 儲存　SUB 減　JPZ(jump zero)　JUMP 跳往

指令週期 Instruction cycle: Registers 寄存器

程序計數器 (PC) IP	貯存下一個將被執行的指令的地址。 program counter
指令寄存器 (IR)	貯存取自主存貯器的正要執行的指令。 instruction register
指令譯碼器 decoder	用來解釋貯存在指令寄存器 IR 的現行指令的邏輯電路。
地址寄存器 (MAR)	貯存指令或數據的地址。 memory address register
數據寄存器 (MDR)	貯存指令或數據。 memory data register
累加器 (ACC) AX	貯存運算的數據或結果 accumulator
程序狀態寄存器 (PSR/SR)	貯存運算後 ACC 結果/狀態 program status register (overflow 滿溢, sign 正負數, zero 零, parity, carry 進位)

NV UP EI PL NZ NA PO NC (01110000)

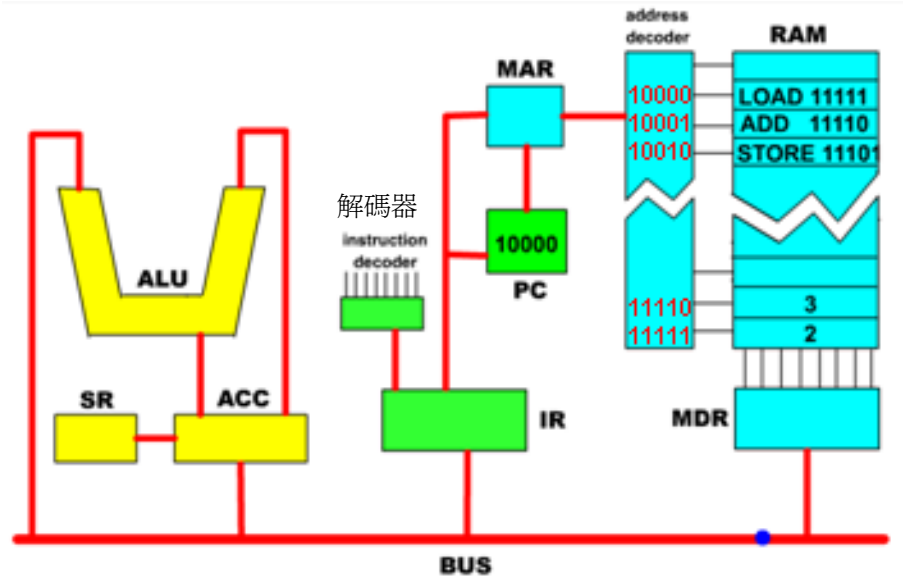


CU 包括： 指令寄存器 IR、程序計數器 PC/IP、指令譯碼器、MAR、MDR。

ALU 包括： 累加器 ACC、程序狀態寄存器 PSR

http://content.edu.tw/senior/computer/ks_ks/et/cpu/index.htm

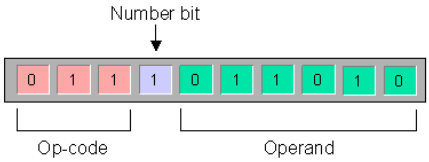
http://www.c-jump.com/CIS77/CPU/InstrCycle/lecture.html



address	instruction/data
10000	LOAD 11111
10001	ADD 11110
10010	STORE 11101
11101	
11110	3
11111	2

cpuoperation.swf

http://courses.cs.vt.edu/csonline/MachineArchitecture/Lessons/CPU/



指令：*操作碼* *操作數*

Op-code 操作碼	*Mnemonic 助記碼*	*Function*	*Example*
001	LOAD	Load 載入 (ACC=memory[1000])	LOAD 1000
010	STORE	Store 儲存至 (memory[0008]=ACC)	STORE 0008
011	ADD	Add 加 (ACC+=5)	ADD #5
100	SUB	Subtract 減 (ACC−=1)	SUB #1
101	EQUAL	If operand = Acc, skip next instruction	EQUAL #20
110	JUMP	Jump to 跳至 (PC=6)	JUMP 0006
111	HALT	Stop execution 停止執行	HALT

Instruction-cycle-sum.swf: $x = 2$, $y = 5$, $z = x + y$

#	*Machine code*	*Assembly code*	*Description*
0	001 1 000010	LOAD #2	Load 載入 ACC=2
1	010 0 001101	STORE 0013	Store 儲存至 memory[13]=ACC (x=ACC)
2	001 1 000101	LOAD #5	Load 載入 ACC=5
3	010 0 001110	STORE 0014	Store 儲存至 memory[14]=ACC (y=ACC)
4	001 0 001101	LOAD 0013	Load 載入 ACC=memory[13]) (ACC=x)
5	011 0 001110	ADD 0014	Add 加 ACC+=memory[14])(ACC+=y)
6	010 0 001111	STORE 0015	Store 儲存至 memory[15]=ACC (z=ACC)
7	111 0 000000	HALT	Stop execution 停止執行

Instruction-cycle-count.swf

#	*Machine code*	*Assembly code*	*Description*
0	001 1 000101	LOAD #5	ACC=5
1	010 0 001111	STORE 0015	memory[0015]=ACC (z=5)
2	001 1 000000	LOAD #0	ACC=0
3	101 0 001111	EQUAL 0015	if (ACC==memory[0015]) PC++; // skip instruction 4
4	110 1 000110	JUMP 0006	PC=6
5	111 0 000000	HALT	Stop execution
6	011 1 000001	ADD #1	ACC+=1
7	110 1 000011	JUMP 0003	PC=3

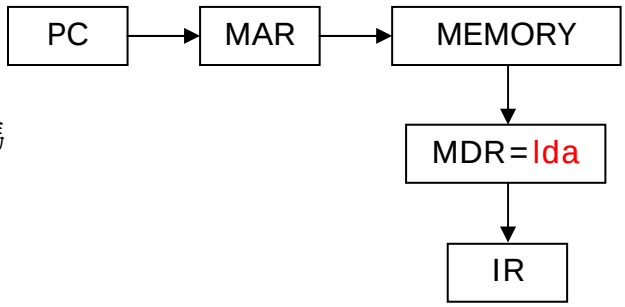
http://www.emu8086.com/assembler_tutorial/tutorials.html

C code: 程式碼

```
n = 10;
n = n+23;
```

6502 assembler: 組合語言(8 bit)

```
0500   lda #23     Load 載入 value into the accumulator 累加器(ACC=23)
0502   add $2043   Add 加 data found at memory 2043h (ACC=ACC+n)
0505   sta $2043   Store 儲存於 at memory 2043h (n=ACC)
...
2043   10          (int n=10)
```



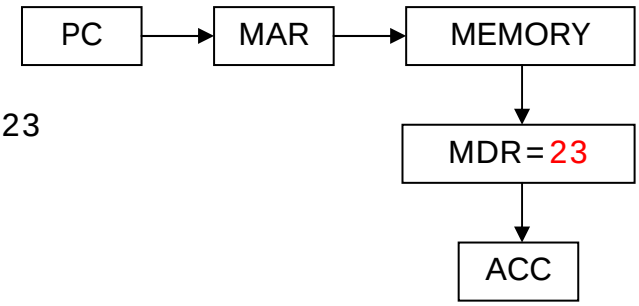
As 6502 machine code in memory: 機器碼

```
9a 23 8a 43 20 84 43 20 ..
```

lda #23 (2 cycles)

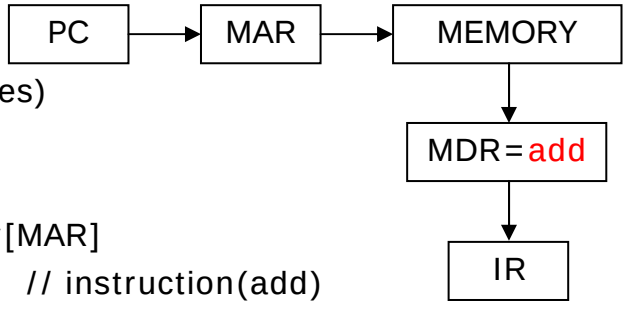
```
Fetch:  PC=0500          // address of next instruction
    MAR=PC               // memory address register
    MDR=memory[MAR]      // memory data register
    IR=MDR               // instruction(lda)
    PC++
```

0500	lda #23
0502	add $2043
0505	sta $2043
2043	10




```
Decode:    operand (#23) needed
    MAR=PC               // PC=0501
    MDR=memory[MAR]      // operand #23
    PC++
```

```
Execute:
    ACC=MDR              // #23
    PSR updated
```



add $2043 (4 cycles)

```
Fetch:  PC=0502
    MAR=PC
    MDR=memory[MAR]
    IR=MDR           // instruction(add)
    PC++
```

0500	lda #23
0502	add $2043
0505	sta $2043
2043	10

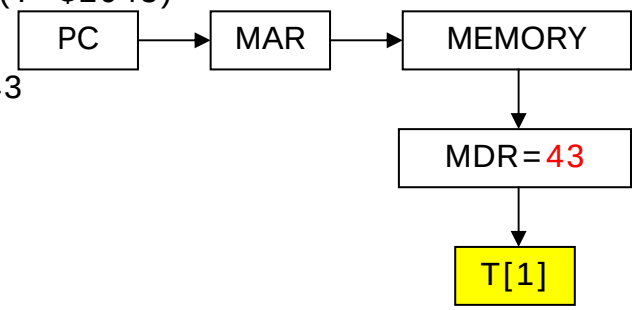
Decode: next 2 bytes are addresses (T=$2043)

MAR=PC

MDR=memory[MAR] // address 43

T[1]=MDR

PC++

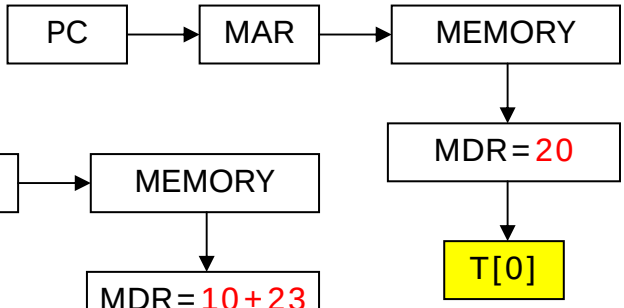


MAR=PC

MDR=memory[MAR] // address 20

T[0]=MDR

PC++



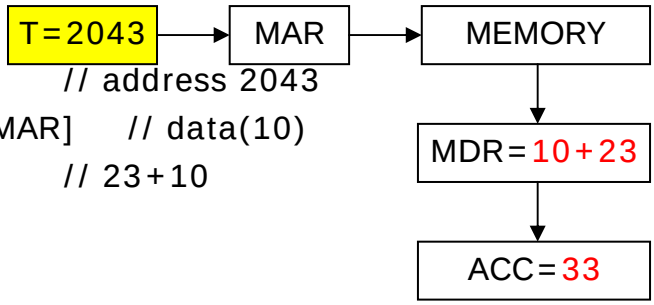
Execute:

MAR=T // address 2043

MDR=memory[MAR] // data(10)

ACC=ACC+MDR // 23+10

PSR updated



sta $2043 (4 cycles)

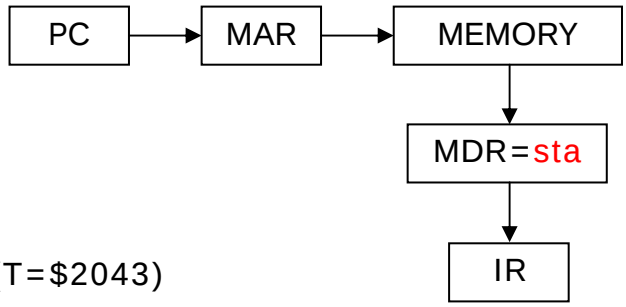
Fetch: PC=0505

MAR=PC

MDR=memory[MAR]

IR=MDR // instruction(sta)

PC++



Decode: next 2 bytes are addresses (T=$2043)

MAR=PC

MDR=memory[MAR] // address 43

T[1]=MDR

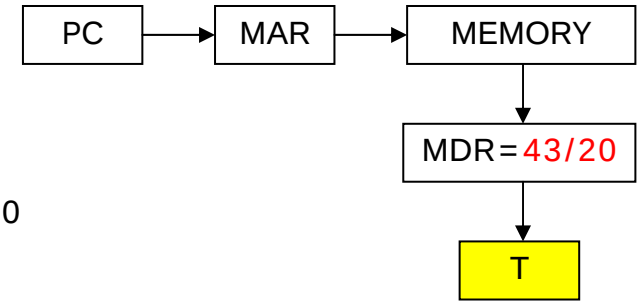
PC++

MAR=PC

MDR=memory[MAR] // address 20

T[0]=MDR

PC++

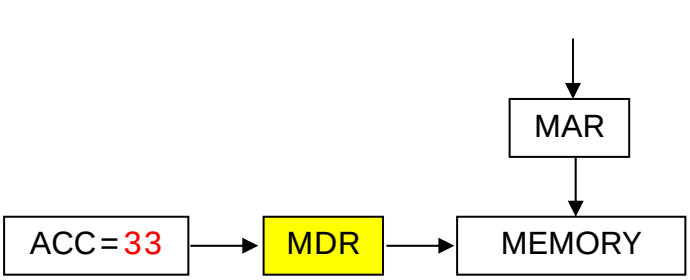


Execute:

MAR=T // address 2043

MDR=ACC

memory[MAR]=MDR



1. Which of the following registers is NOT in the Control Unit? 下列哪些寄存器不在控制部件(CU)內?

(1) IR
(2) PC
(3) MDR
(4) PSR

A. (3) only
B. (4) only
C. (1) and (2) only
D. (2) and (3) only
E. (3) and (4) only

2. What is the minimum number of bits required to represent 1024 different operation codes? 最少需要多少位元去代表 1024 不同的操作碼?

A. 8 B. 9 C.10 D.11 E. 12

3. In an instruction fetch cycle, instruction is read from P to Q. Which of the following combinations is correct? 在指令提取週期，指令由 P 搬到 Q。

	P =	Q =
A.	main memory	ALU
B.	PC	main memory 主記憶體
C.	IR	ALU
D.	main memory	IR
E.	MAR	MDR

4. In an instruction fetch cycle, there is address flow from P to Q directly. Which of the following combinations is correct? 指令提取週期，地址由 P 直接搬到 Q。

	P =	Q =
A.	MDR	ALU
B.	main memory	MDR
C.	PC	main memory 主記憶體
D.	PC	MAR
E.	MAR	MDR

5. In an instruction fetch cycle, there is data flow from P to Q directly. Which of the following combinations is correct? 指令提取週期，數據由 P 直接搬到 Q。

	P =	Q =
A.	MDR	ALU
B.	main memory	MAR
C.	PC	main memory 主記憶體
D.	PC	MAR
E.	MAR	MDR

1. (a) Name the following registers: 寫出以下寄存器的名稱。

PC, IR, MAR, MDR, PSR, AX(ACC)

(b) What do they store during the instruction cycle? 每個寄存器貯存些什麼?

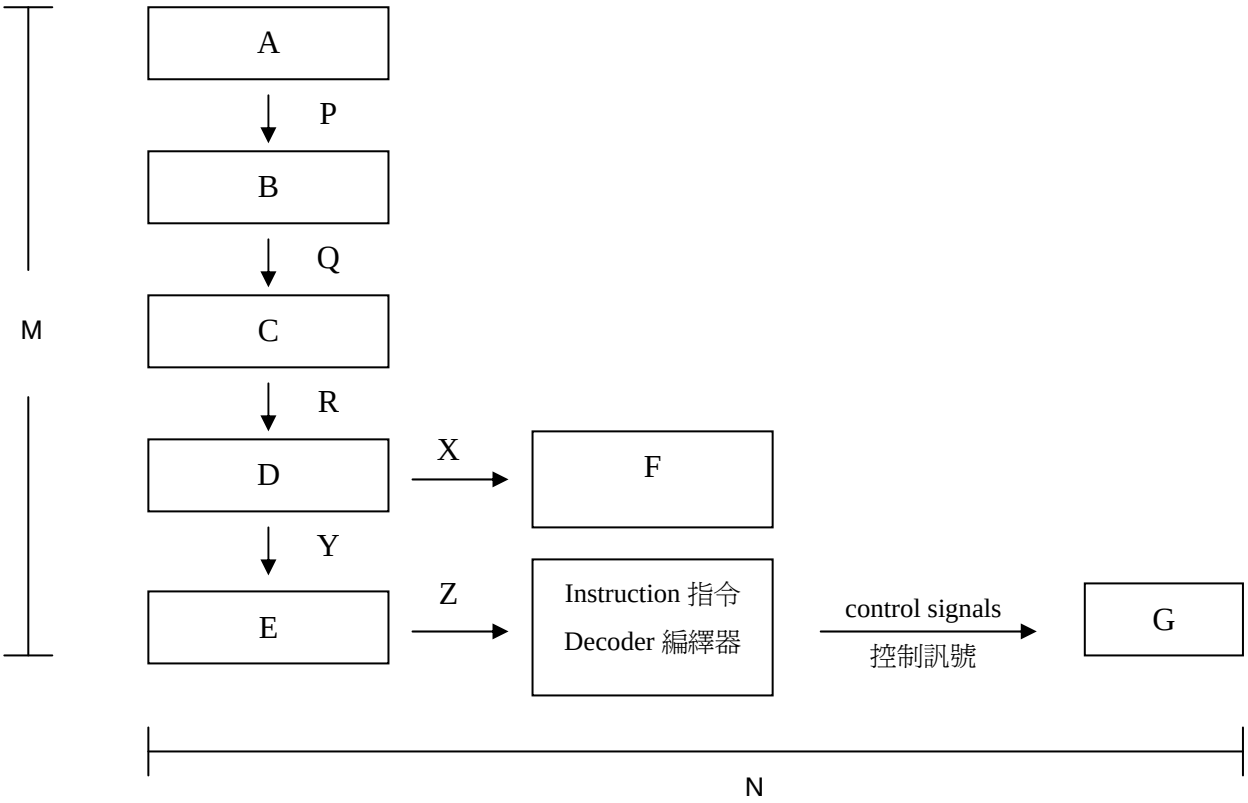
2. (a) What are the 3 stages of an instruction cycle? 列出「指令週期」的三個階段

Describe how instructions are being fetched, decoded and executed and how registers are involved in the instruction fetch and execution cycles.

試描述在指令週期中，指令如何被提取、解碼和執行；同時寫下寄存器所担當的角色。

(b) Label the following diagram which depicts the instruction cycle 指令週期 (填圖)

A		P	
B		Q	
C		R	
D		X	
E		Y	
F		Z	
G	電腦各部件 various parts of computer		
M		N	



suggested solution

Section A:

Section B:

Instruction cycle 指令週期:

1	Fetch an instruction from the main memory into IR 由主記憶體取出指令
2	Decode the instruction to control signals; 編譯(指令寄存器內的)現行指令
3	Execute the instruction 執行指令 by sending the control signals to various parts of the computer system

By automatically increasing the content in the PC, the instruction cycle is repeated.自動增加 PC 值

An instruction cycle 指令週期 can also be divided into

(a) instruction fetch cycle 提取週期

1	the PC sends the address of the instruction to MAR 程序計數器 PC 把將要執行指令所在地址送到 MAR 地址寄存器
2	the MAR sends address signals to the main memory through address bus MAR 將地址訊號 (經地址總線) 送到主記憶體
3	the instruction is read from address specified to MDR through data bus 讀入指令到 MDR 數據寄存器 (經數據總線)
4	the MDR sends the instruction to the IR 把指令由 MDR 數據寄存器送到 IR 指令寄存器
5	PC is increased by one (or updated so that it points to the next instruction to be executed) PC 值自動加一，指向下一句被執行指令

(b) decode 解碼

6	IR sends the instruction to the instruction decoder IR 傳送指令到指令譯碼器
7	the instruction decoder decodes the instruction to control signals 指令譯碼器 解釋/編譯 指令成爲 控制訊號

(c) instruction execution cycle 執行週期

8	control signals are sent to various parts of the computer system through control bus 控制訊號 (經控制總線) 被傳送到電腦各部分
9	執行有關指令 perform arithmetic and/or logical operations 進行算術、邏輯運算 update PSR if necessary 更新 PSR 狀態 load data into ACC or store value to RAM 載入或 貯存 數據