EE 3480

Vi Xử Lý

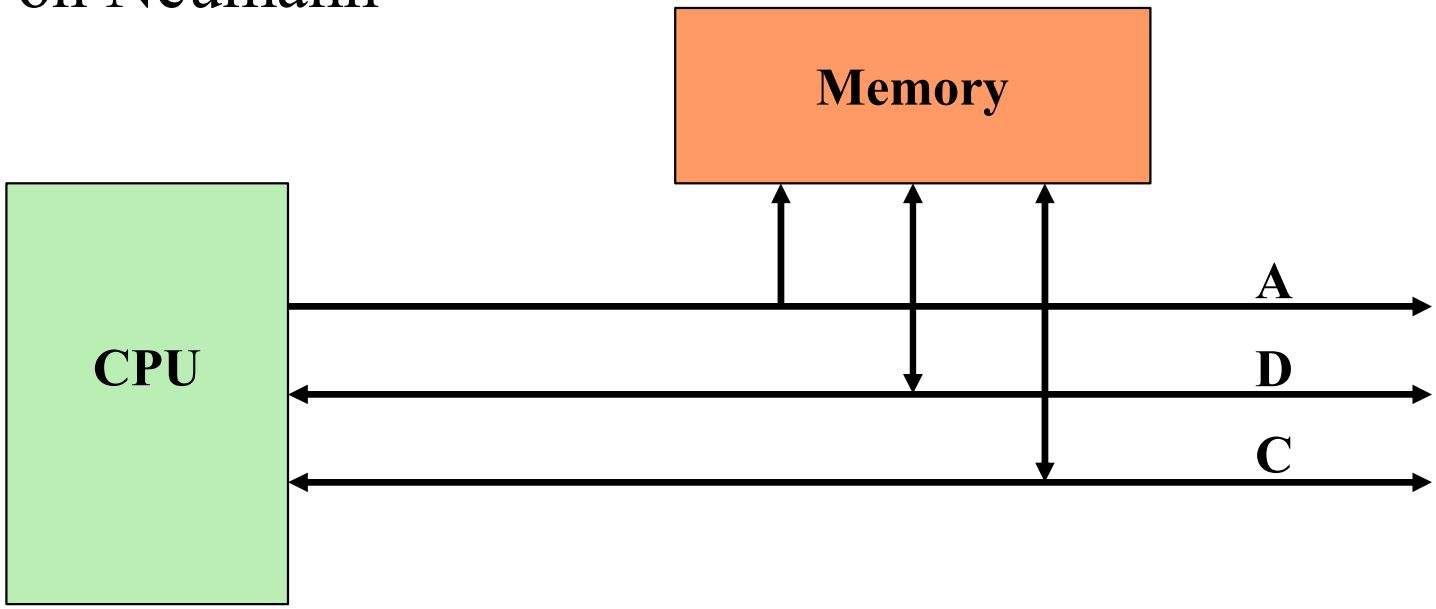
Microprocessor - MicroController

Nguyen Tuan Ninh - 3I - SEEE - HUST

ninh.nguyentuan@hust.edu.vn

Kiến trúc hệ VXL

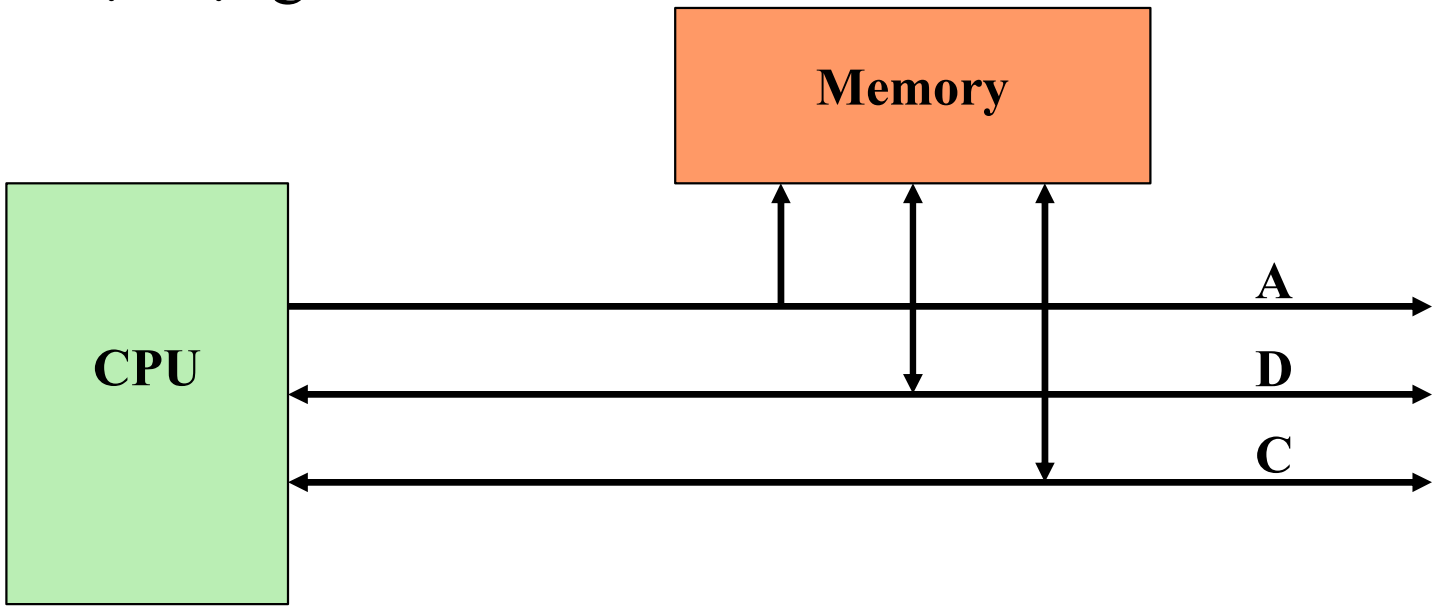
- Kiến trúc Von Neumann



- CPU: Bộ vi xử lý trung tâm: thực hiện lệnh và công việc
- Memory: Bộ nhớ lưu trữ lệnh, chương trình
- A (Address bus): Quyết định độ lớn miền không gian nhớ
- D (Data bus): 8 bit (D0 – D7)
- C (Control bus): Các tín hiệu Rd, Wr, Int, …

Kiến trúc hệ VXL

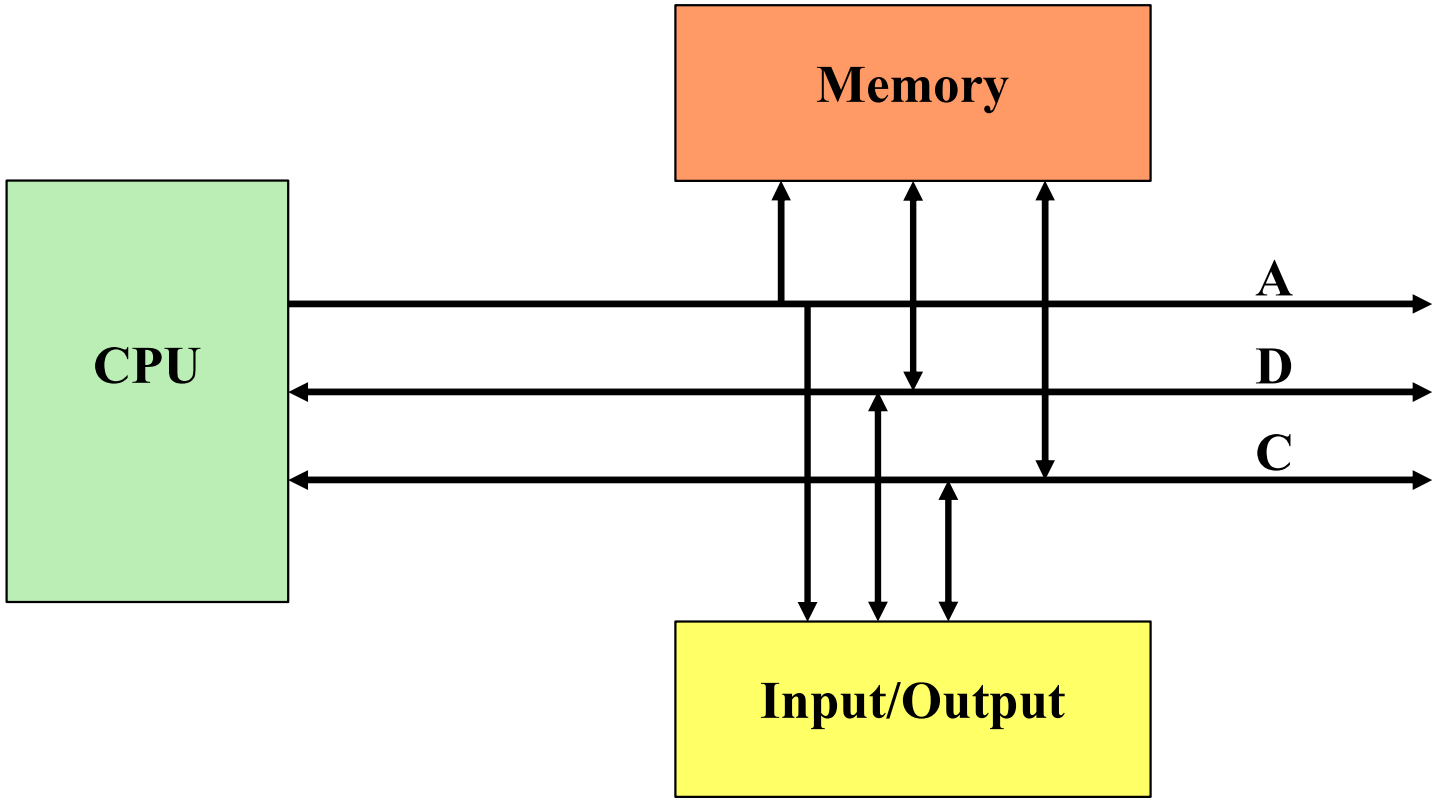
- Nguyên tắc hoạt động



- Mỗi CPU thực hiện một tập lệnh hữu hạn
- Tập hợp các lệnh trở thành 1 chương trình, lưu dưới dạng mã nhị phân trong bộ nhớ chương trình.
- CPU thực hiện lệnh 1 cách tuần tự
- Để thực hiện lệnh rẽ nhánh có điều kiện, CPU căn cứ vào các cờ (flag)
- CPU truy cập đến ô nhớ, thiết bị ngoại vi thông qua nguyên tắc địa chỉ

Kiến trúc hệ VXL

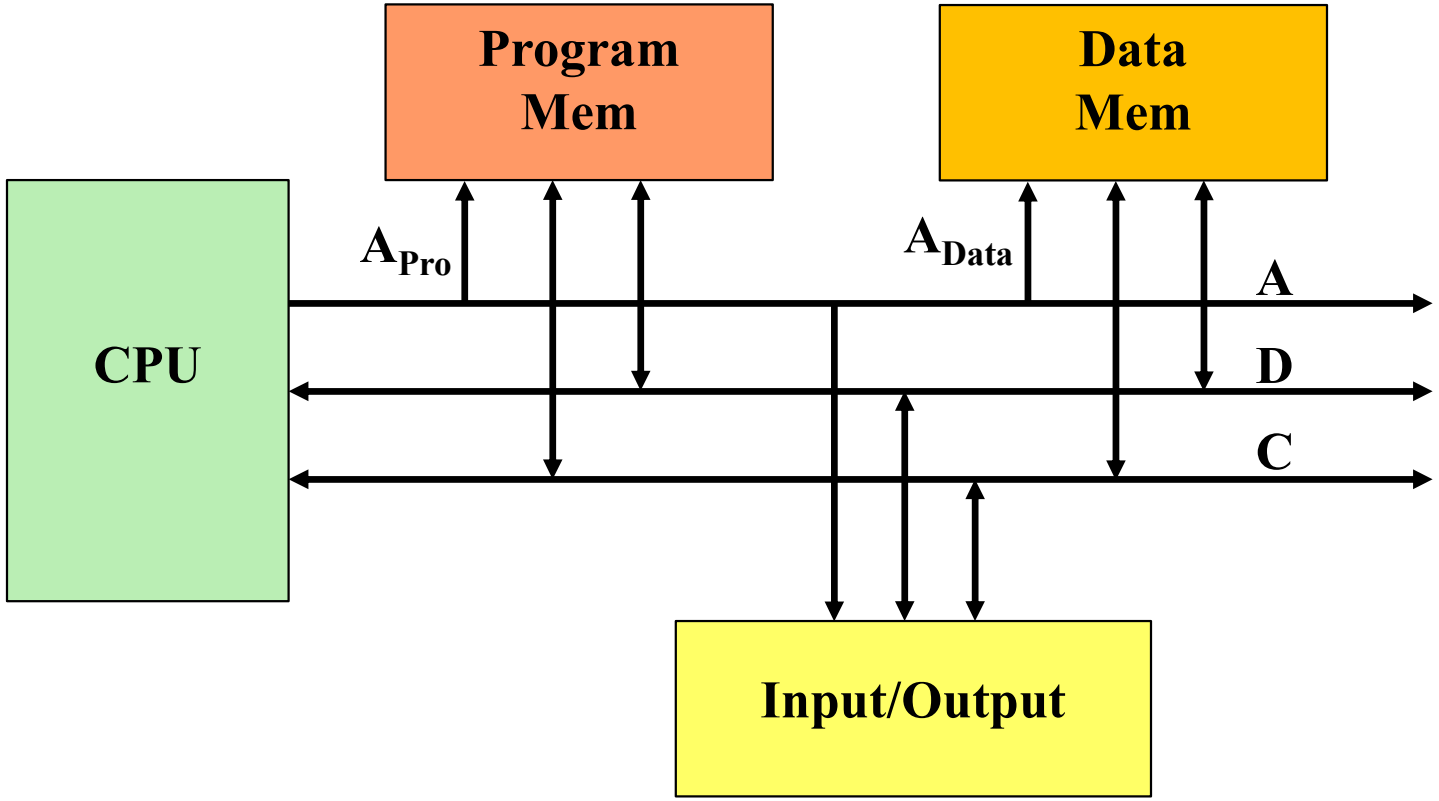
- Kiến trúc Von Neumann mở rộng



  - Input/Output (I/O): Thiết bị ngoại vi

Kiến trúc hệ VXL

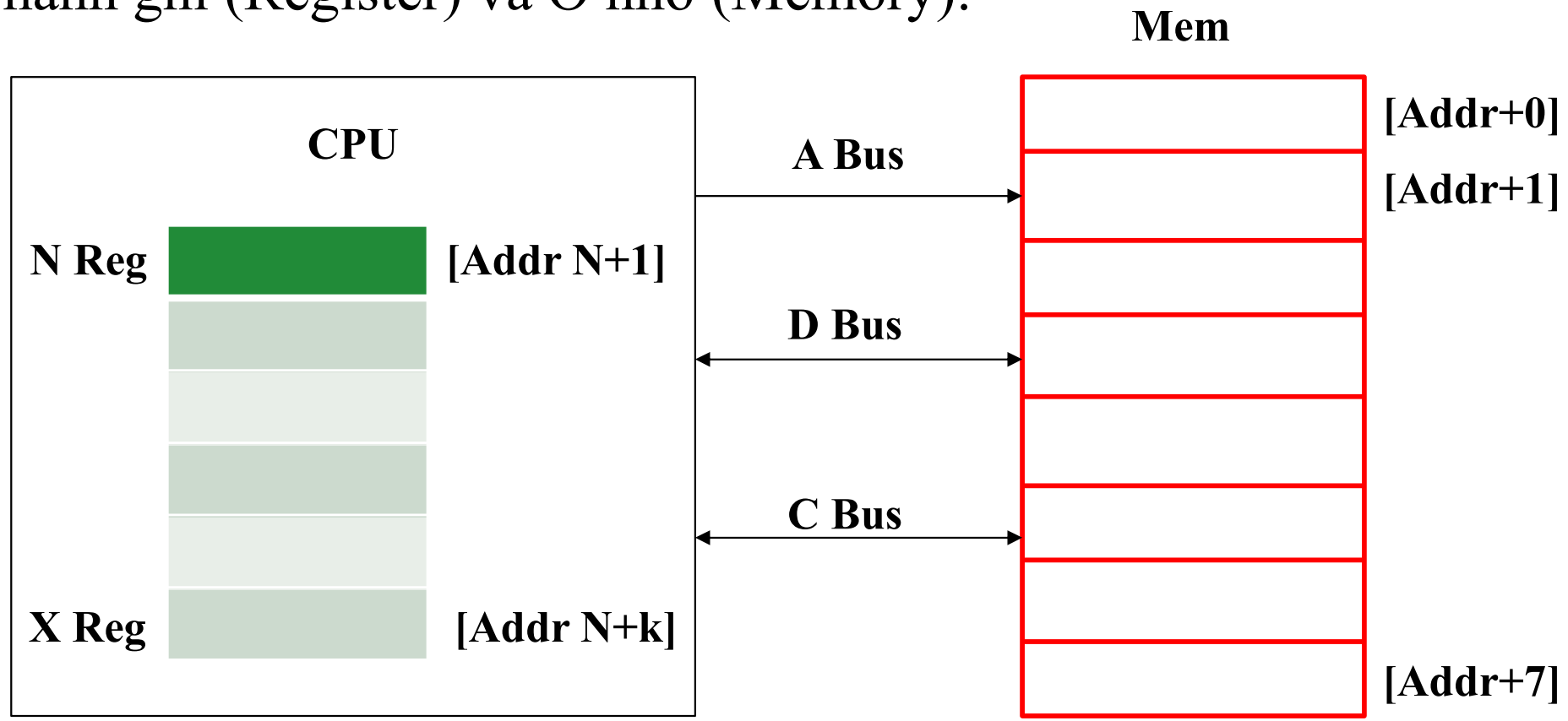
- Kiến trúc Harvard



- Tách Bộ nhớ chương trình (Program Mem) và Bộ nhớ dữ liệu (Data Mem)

Nguyên tắc thực hiện 1 lệnh

- Đọc tài liệu mcs51-manual(trang 2-21 …)
  - Bảng 10, 8051 instruction set summary: Mô tả các lệnh, số bit cần dùng
  - Bảng 11: Instruction opcodes in Hexadecimal Orders: bảng mã lệnh
  - Trang 2-28: Giải thích cụ thể các lệnh và cung cấp các ví dụ
- Thanh ghi (Register) và Ô nhớ (Memory):



Nguyên tắc thực hiện 1 lệnh

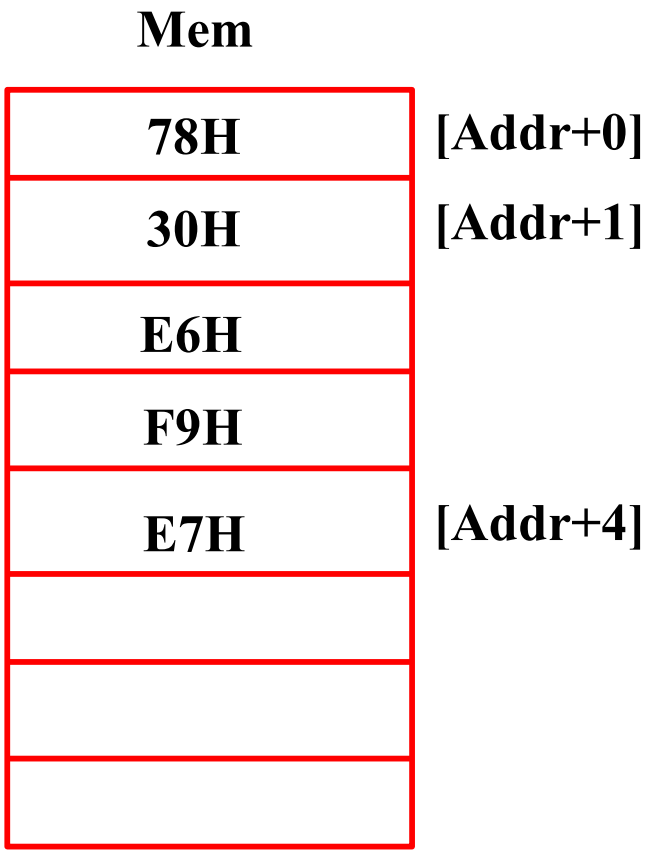
- Cấu trúc câu lệnh:
- Các chế độ địa chỉ:
  - Câu lệnh chung : Mov OP1, OP2; #

    mã lệnh — toán hạng — chú thích
  - Câu lệnh có 1 toán hạng : INC OP1
  - Câu lệnh chỉ có mã lệnh : NOP
- Các chế độ địa chỉ (Addressing mode):
  - Direct Addressing Add A, 7FH
  - Indirect Addressing Add A,@R0
  - Register Addressing Add A,R7
  - Immediate Constants Add A,#127
  - Indexed Addressing

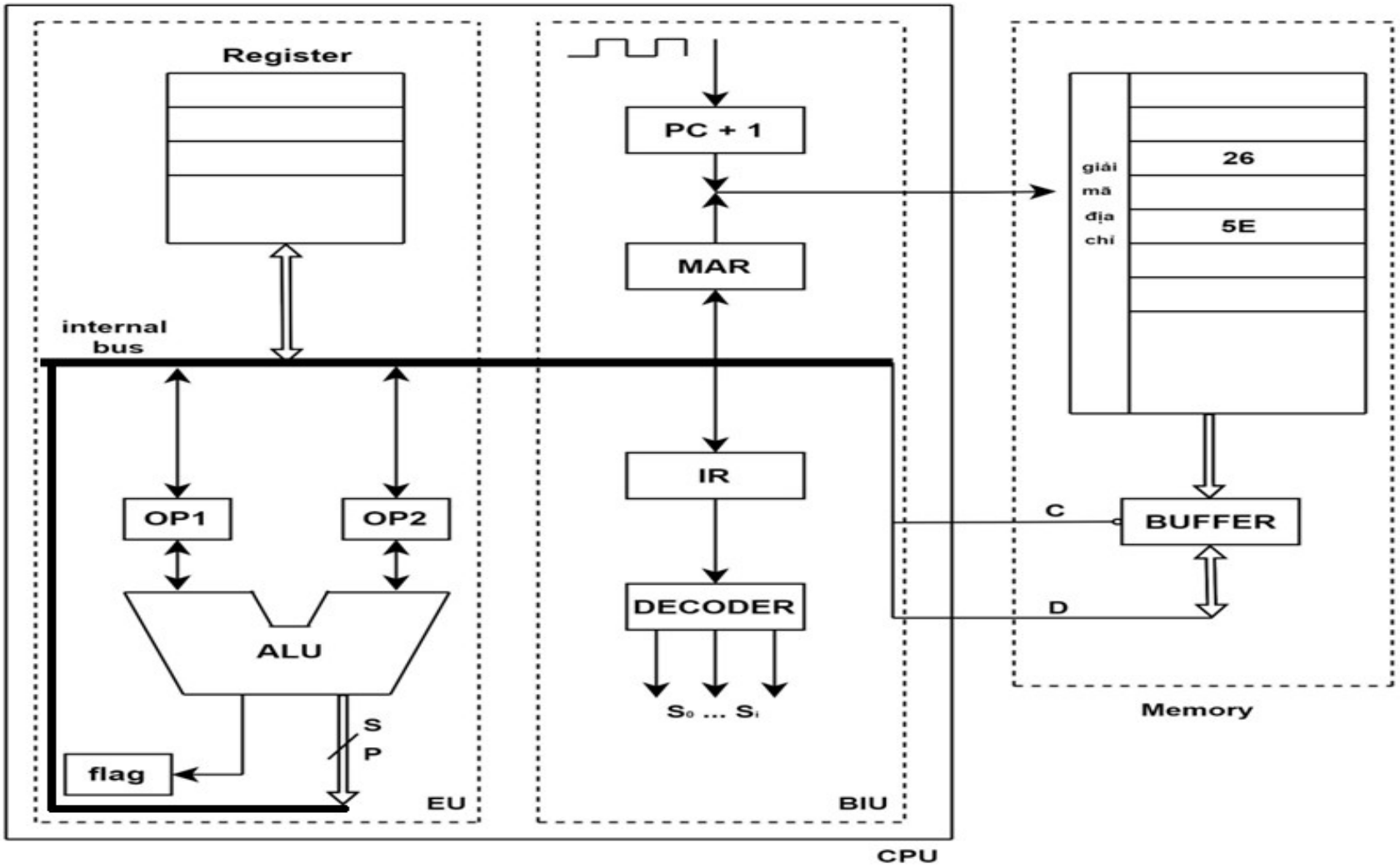
Nguyên tắc thực hiện 1 lệnh

- Phân tích câu lệnh cơ bản MSC 51:

Mov	R0, #30H;	78H 30H
Mov	A, @R0;	E6H
Mov	R1, A;	F9H
Mov	A, @R1;	E7H



Nguyên tắc thực hiện 1 lệnh

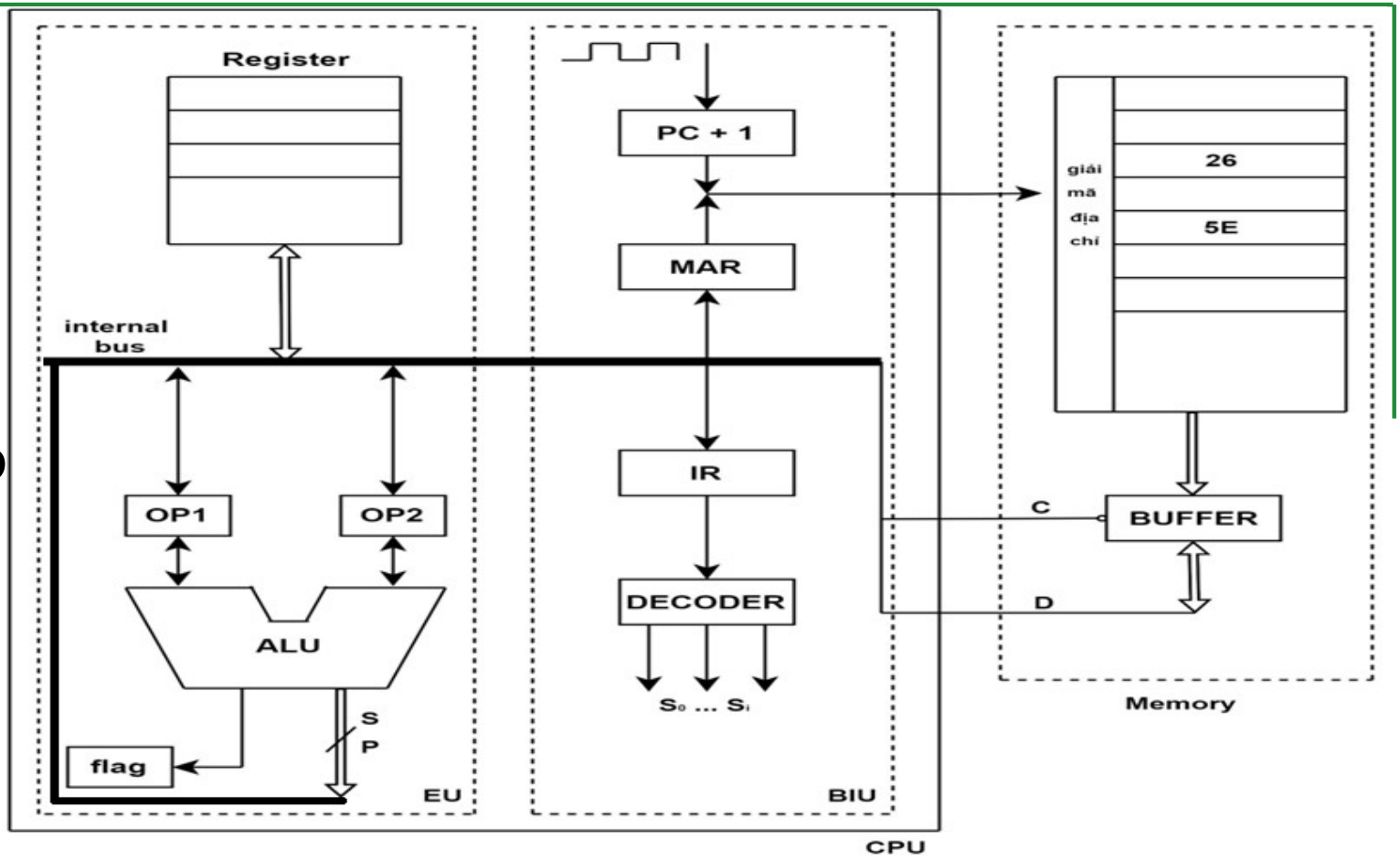


- PC (Program counter)
- MAR: Memory Addr Reg
- IR: Instruction Reg
- ALU: Bộ ALU
- EU: Excution Unit
- BIU : Bus Interface Unit
- Register : Thanh ghi
- Op1, Op2: Toán hạng
- Buffer: Bộ đệm

ninh.nguyentuan@hust.edu.vn

Nguyên tắc thực hiện 1 lệnh

- Power On, đưa giá trị của PC (**0000H**) lên Addr bus , qua giải mạch mã địa chỉ truy cập đến Mem, lấy giá trị đưa vào Buffer.



- CPU gửi tiếp tín hiệu Rd lên Control bus, đưa số liệu lên Databus, đưa vào IR
- Qua giải mã lệnh thì CPU sẽ biết làm gì tiếp theo (tùy vào câu lệnh ở chế độ địa chỉ)

Nguyên tắc thực hiện 1 lệnh

- Giải thích các câu lệnh:

Mov R0, #30H ;78H 30H

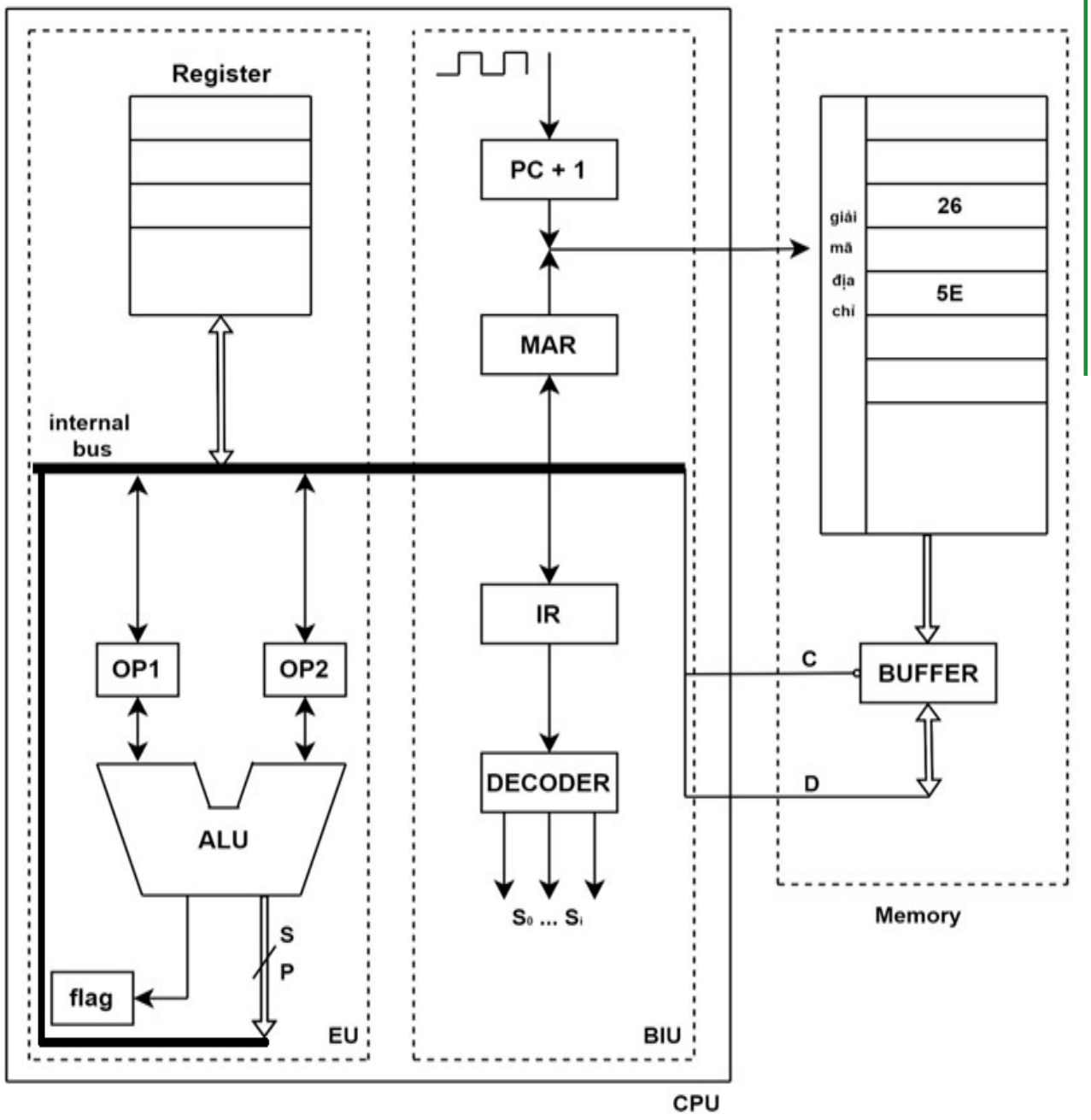
Mov A, @R0 ;E6H

Mov R1, A ;F9H

Mov A, @R1 ;E7H

- Sv giải thích câu lệnh:

Add A,@R0 ; 26H

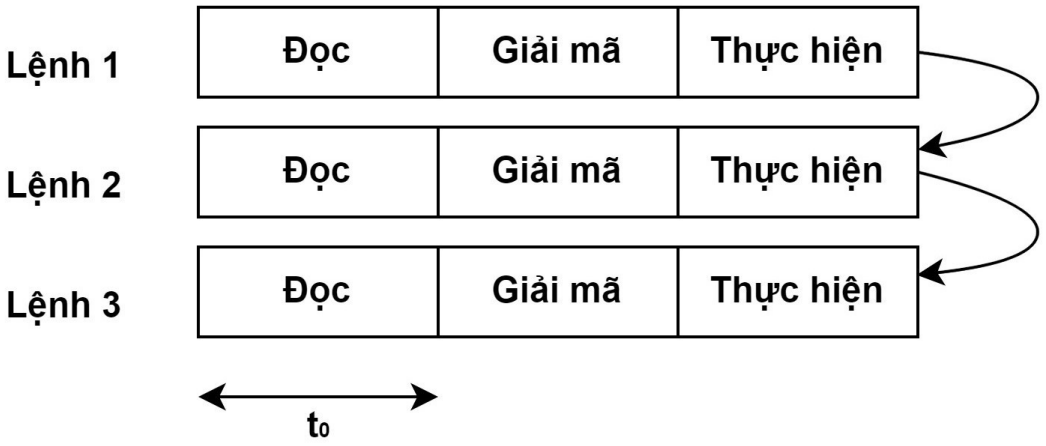


Tăng tốc độ xử lý CPU

- **Vấn đề:** Thực hiện lệnh

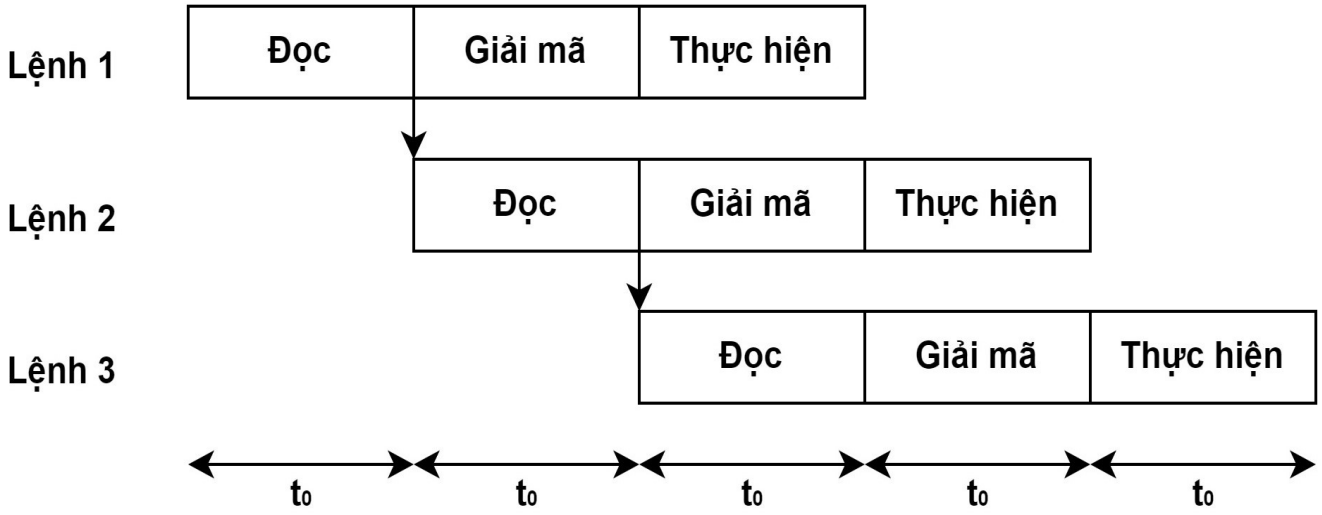
  lệnh 1 -> lệnh 2 -> lệnh 3

Nhiều lệnh hơn thì CPU sẽ tốn thời gian thực hiện CPU chậm.



- **Khắc phục:**

  - C1: tăng tần số xung nhịp (giới hạn bởi công nghệ làm chất bán dẫn -> gần như không thể)



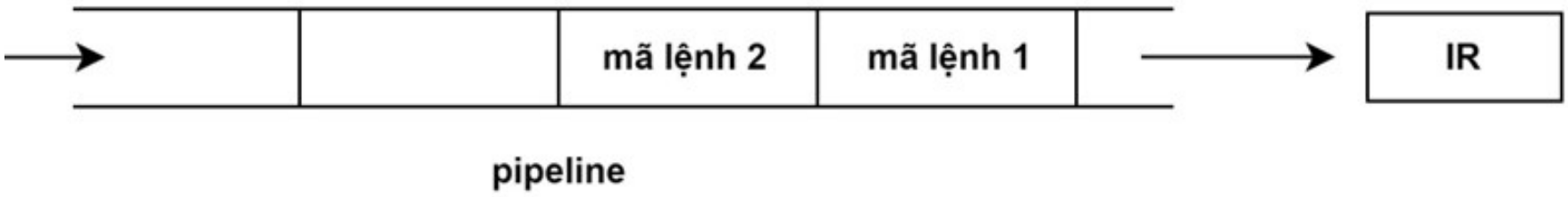
  - C2: chia CPU thành nhiều phần khác nhau (BIU và EU). Trong lúc giải mã và thực hiện lệnh 1 thì đọc tiếp tục lệnh 2 và tương tự đối với lệnh 3 => Chỉ tốn $5.t_0$

PipeLine- Hàng đợi lệnh

- Khi đọc xong lệnh 1, CPU lại tiếp tục đọc lệnh 2, khi đó lệnh 1 vào IR chưa giải mã xong, lệnh 2 cũng được đưa đến IR.

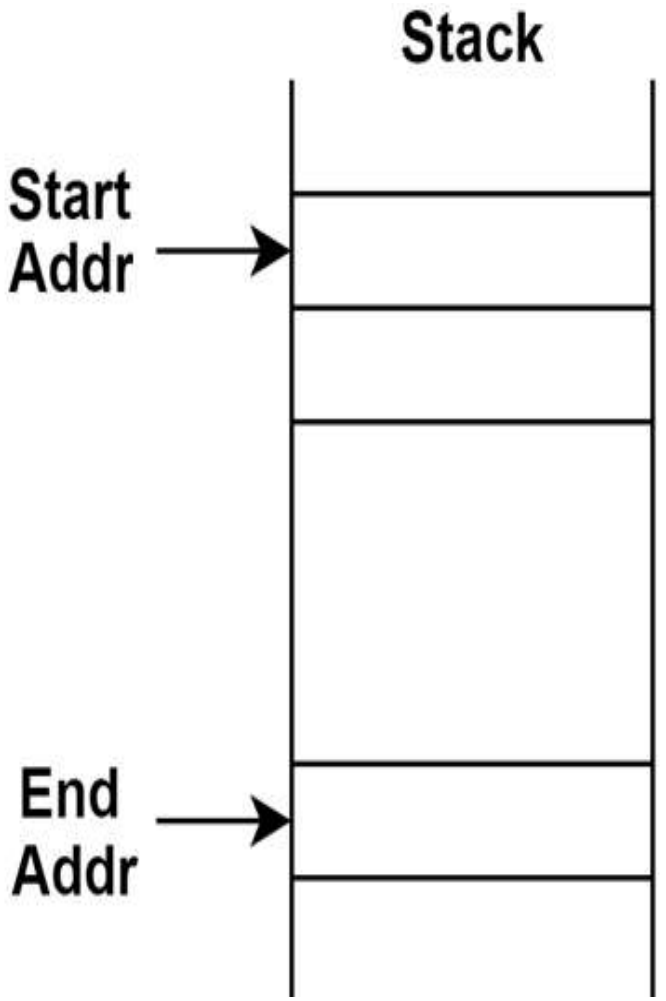
=> Sử dụng hàng đợi lệnh: các lệnh được xếp vào hàng đợi lệnh trước khi được đưa vào IR

- Hàng đợi lệnh hoạt động theo nguyên tắc FIFO (First In First Out)



Stack – Ngăn xếp

- **Định nghĩa:** là một vùng nhớ, thông thường đặt ở trong RAM, dùng để lưu trữ các số liệu tạm thời của một chương trình con hoặc một chương trình phục vụ ngắt
- **Tại sao ngăn xếp đặt ở trong RAM?**



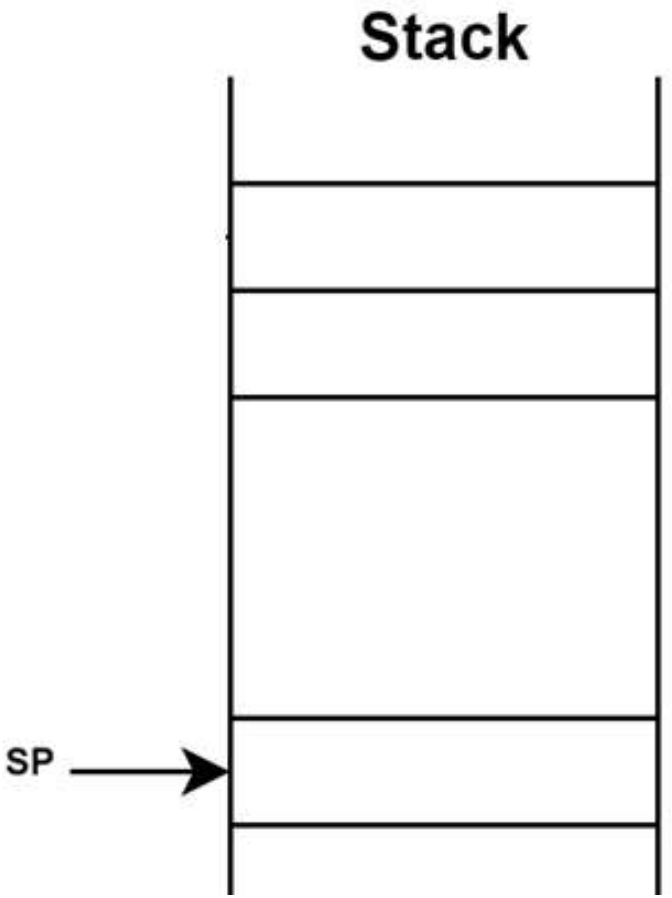
Xét theo tốc độ truy cập:

thanh ghi > RAM > ROM

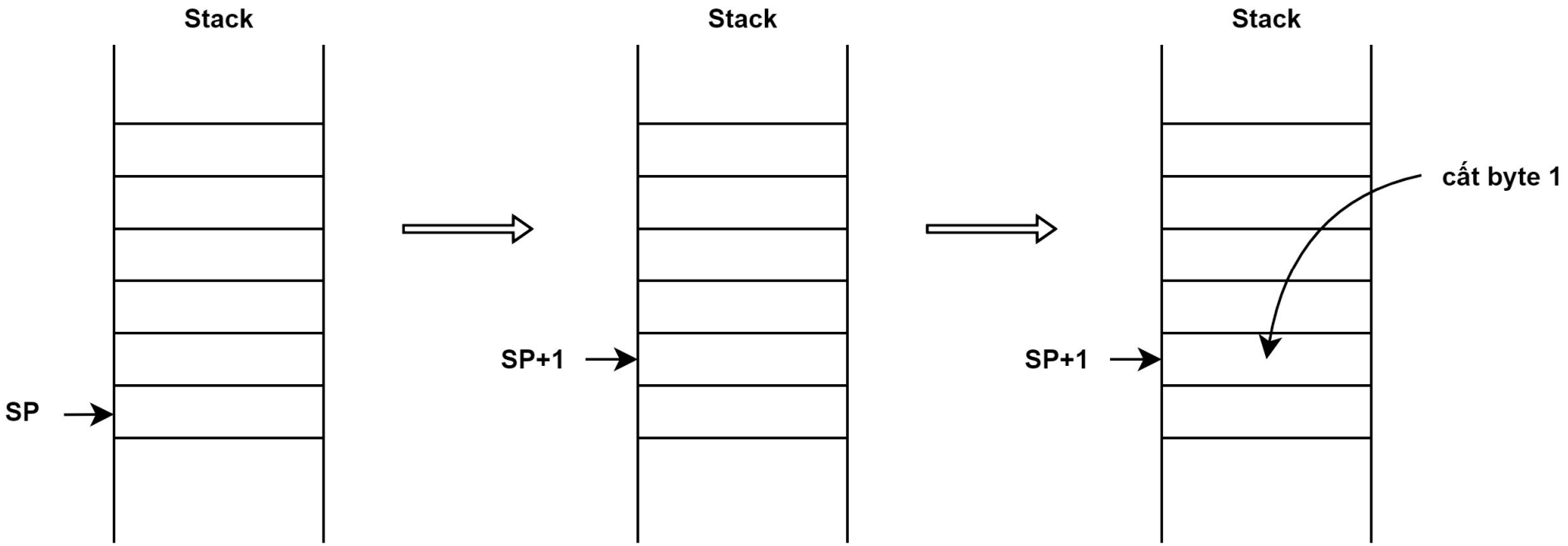
Tuy nhiên vì độ dài chương trình tạm thời có độ dài không xác định (phụ thuộc người viết) nên không dùng được thanh ghi (dung lượng ít và có hạn). Vì vậy, dùng RAM vừa có tốc độ nhanh hơn ROM, vừa có dung lượng lớn hơn và còn có khả năng mở rộng.

Stack – Ngăn xếp

- **SP (Stack Pointer)**: con trỏ ngăn xếp (chỉ vào đỉnh ngăn xếp)
- **Xác định độ rộng ngăn xếp**
  - C1: sử dụng địa chỉ bắt đầu (Start address) và địa chỉ kết thúc (End address)
  - C2: địa chỉ bắt đầu + dung lượng ngăn xếp cần thiết
- **Nguyên tắc hoạt động của ngăn xếp**
  - LIFO (Last In First Out)
  - FIFO (First In First Out)



Stack – LIFO

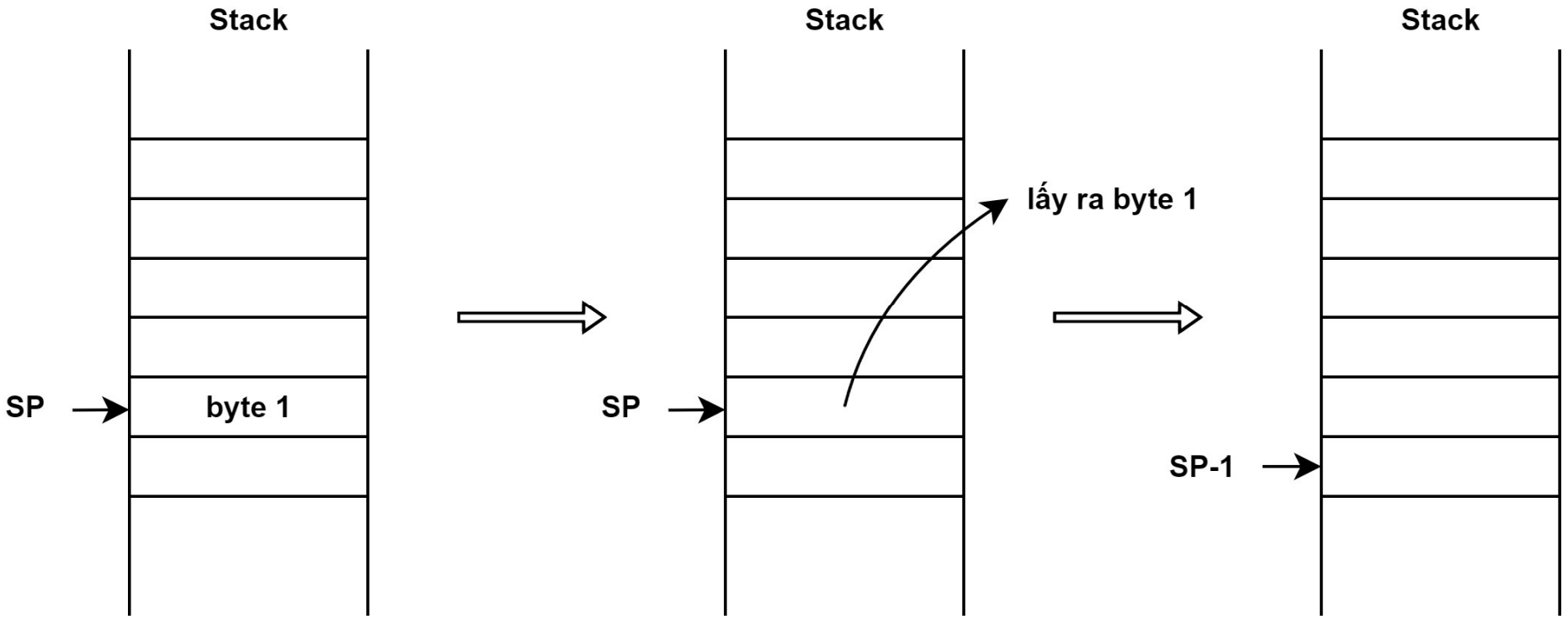


- **Lệnh thao tác ngăn xếp**
  - Push Reg; Lệnh cất vào ngăn xếp

    Sp = Sp +1

    Cất gtri Reg vào ô nhớ chỉ bởi SP

Stack – LIFO



- **Lệnh thao tác ngăn xếp**
  - Pop Reg; Lệnh lấy ra từ ngăn xếp

    Lấy gtri ô nhớ chỉ bởi SP chuyển vào Reg

    Sp = Sp -1

Stack – LIFO

VD: xác định gtrị các thanh ghi, SP và nội dung trong ngăn xếp sau khi thực hiện các lệnh sau

```
MOV     SP,  #2F h
MOV     A,  #1E h
MOV     R0 ,  #E2 h
MOV     R1,  #7D h
ADD     A,  #1C h
PUSH    Acc
PUSH    0
MOV     R0,  R1
POP     1
POP     7
```

Stack – LIFO

Thực hiện 4 lệnh đầu

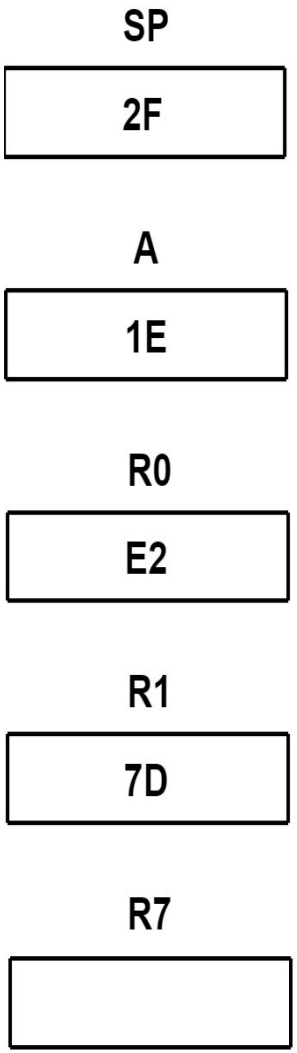
MOV SP , #2F h

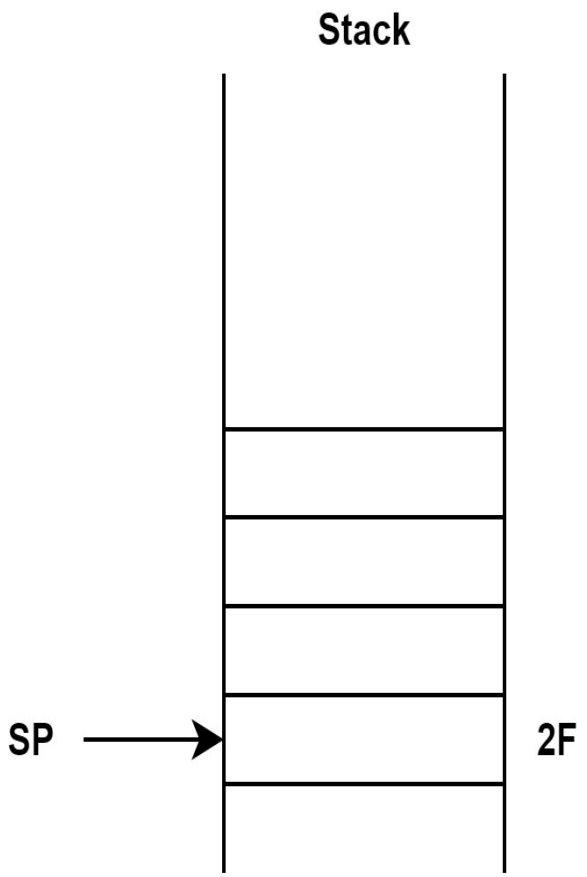
MOV A , #1E h

MOV R0 , #E2 h

MOV R1 , #7D h

Chuyển các giá trị 2F, 1E, E2, 7D lần lượt vào SP, thanh ghi A, R0 và R1



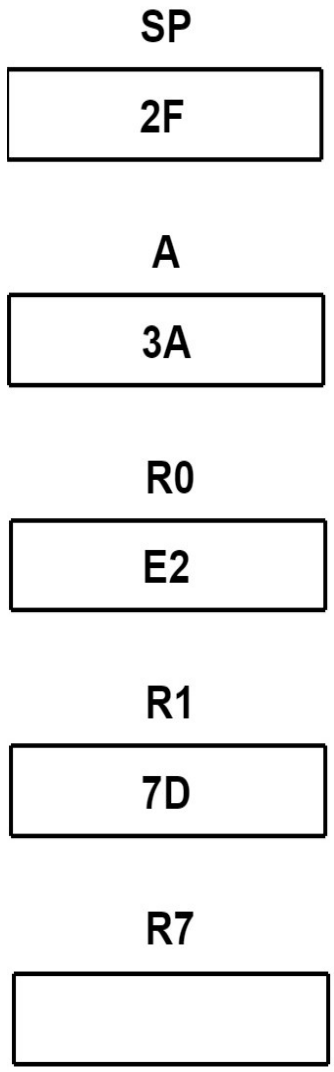


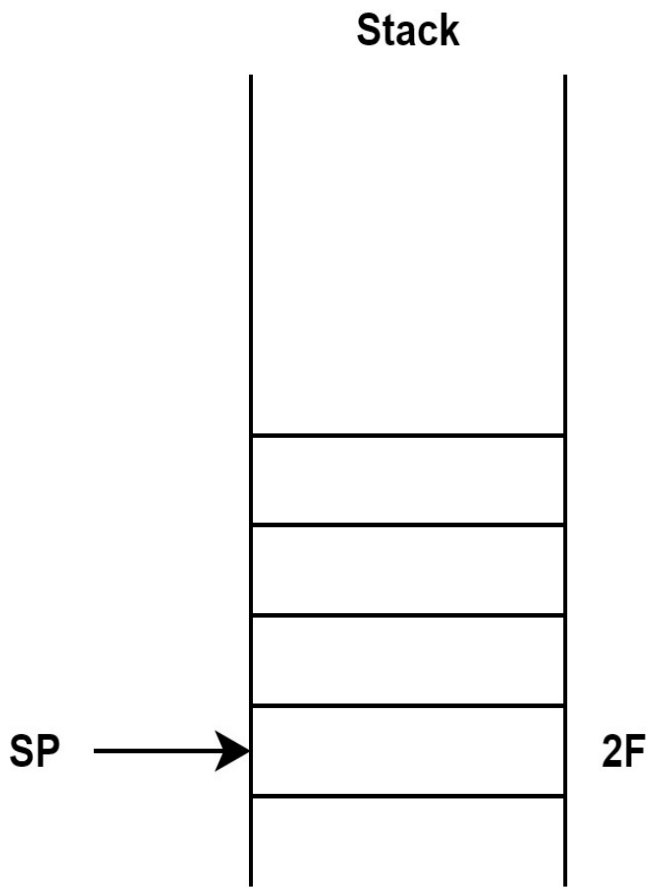
Stack – LIFO

Thực hiện lệnh

ADD A , #1C h

cộng giá trị tại thanh ghi A với giá trị 1C được giá trị 3A rồi lưu lại vào thanh ghi A





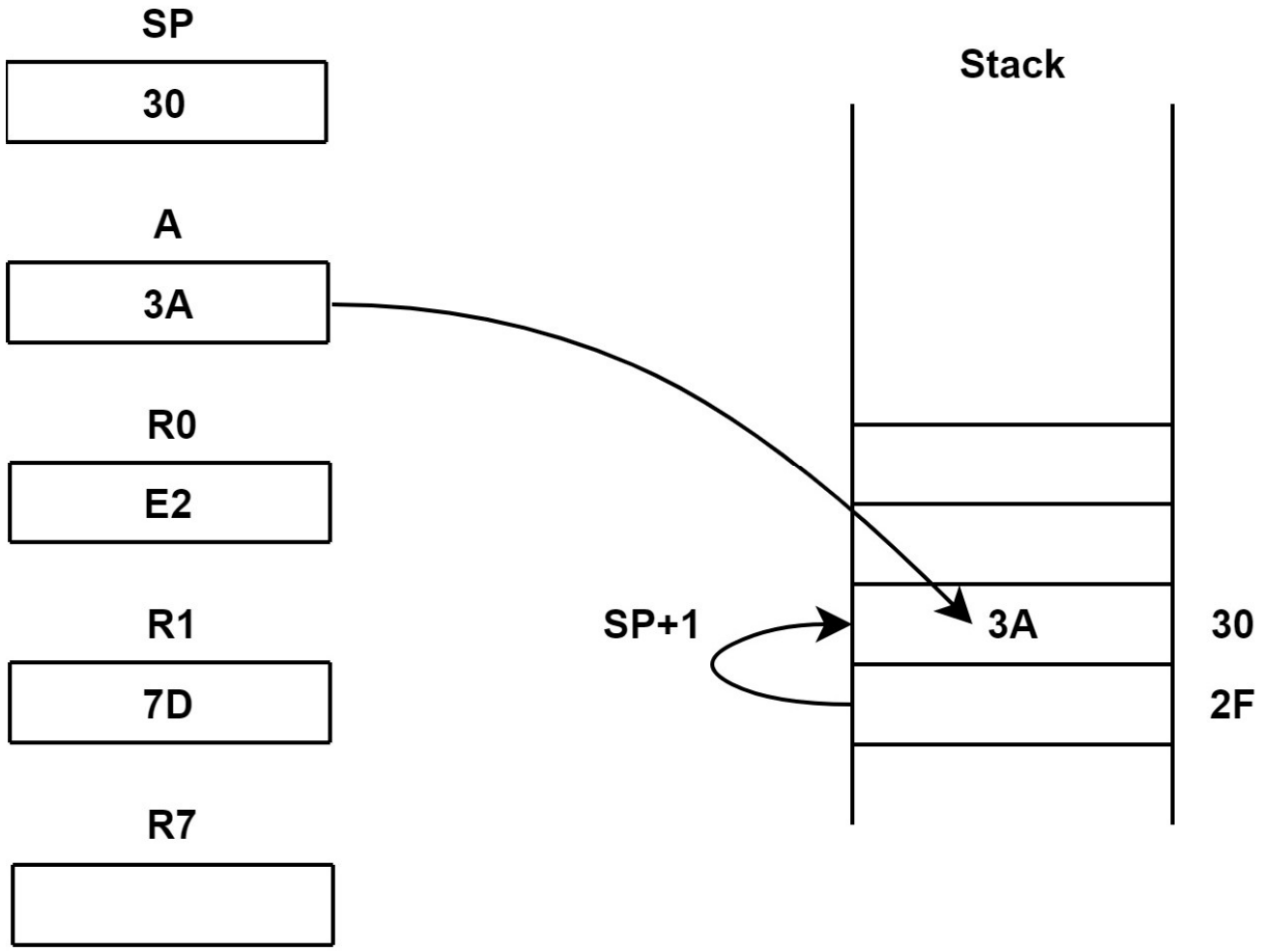
ninh.nguyentuan@hust.edu.vn

Stack – LIFO

Thực hiện lệnh

PUSH Acc

SP = SP+1, cất giá trị tại thanh ghi A vào ngăn xếp đang được trỏ bởi SP

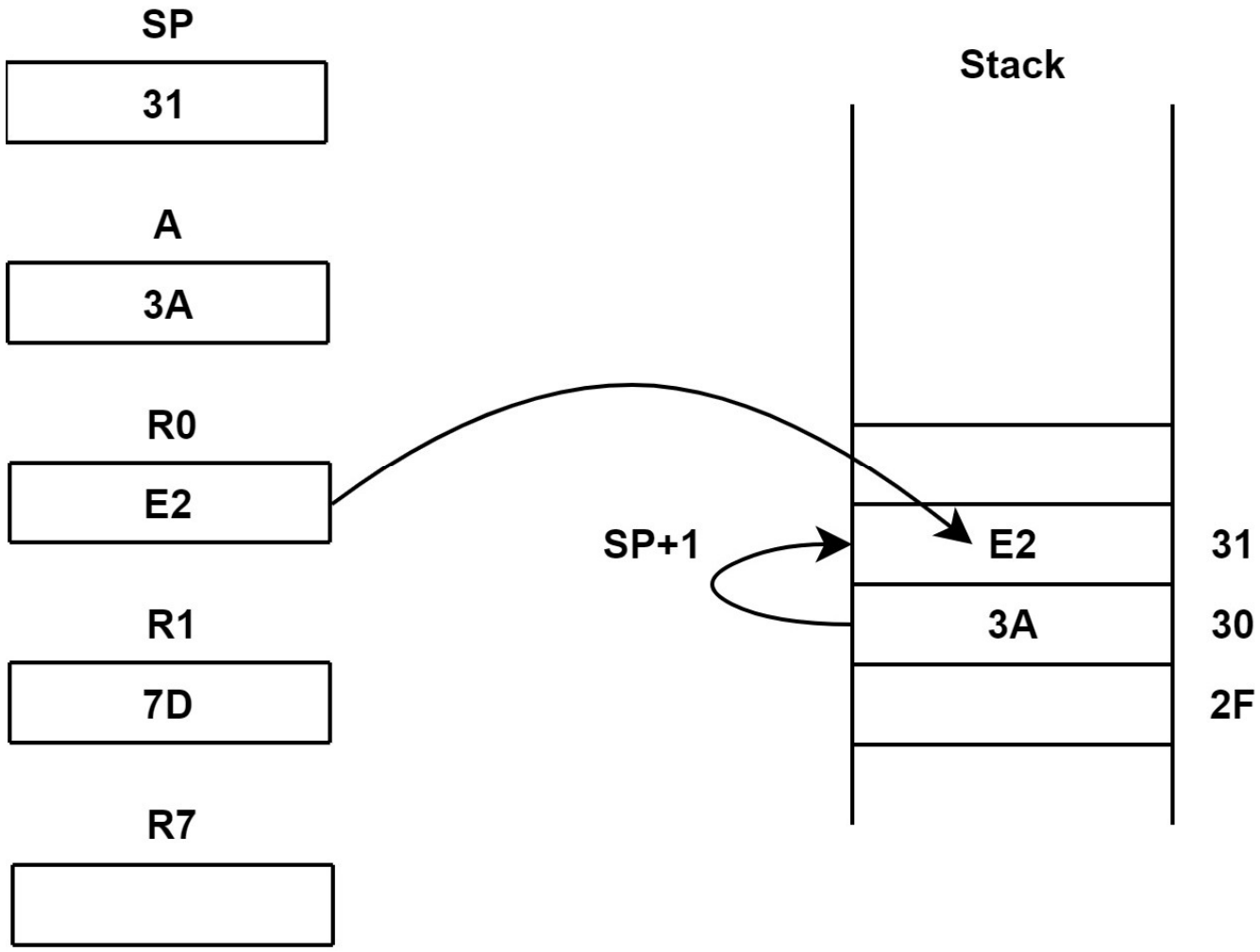


Stack – LIFO

Thực hiện lệnh

PUSH 0

SP = SP+1, cất giá trị tại thanh ghi R0 vào ngăn xếp đang được trỏ bởi SP

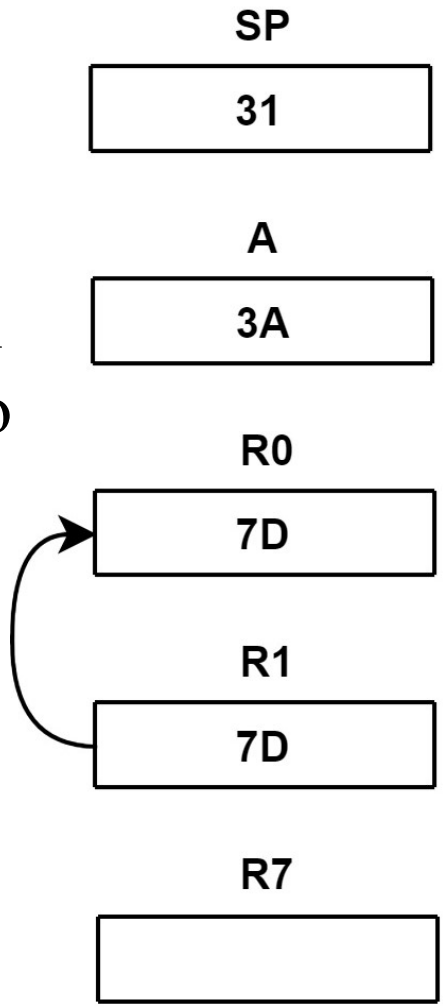


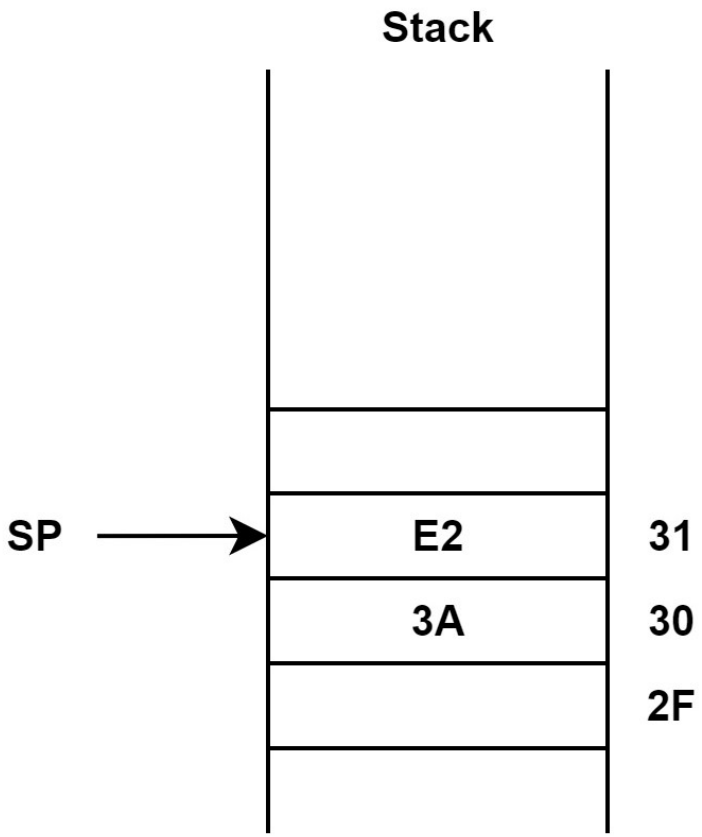
Stack – LIFO

Thực hiện lệnh

MOV R0, R1

chuyển giá trị tại thanh ghi R1 vào thanh ghi R0



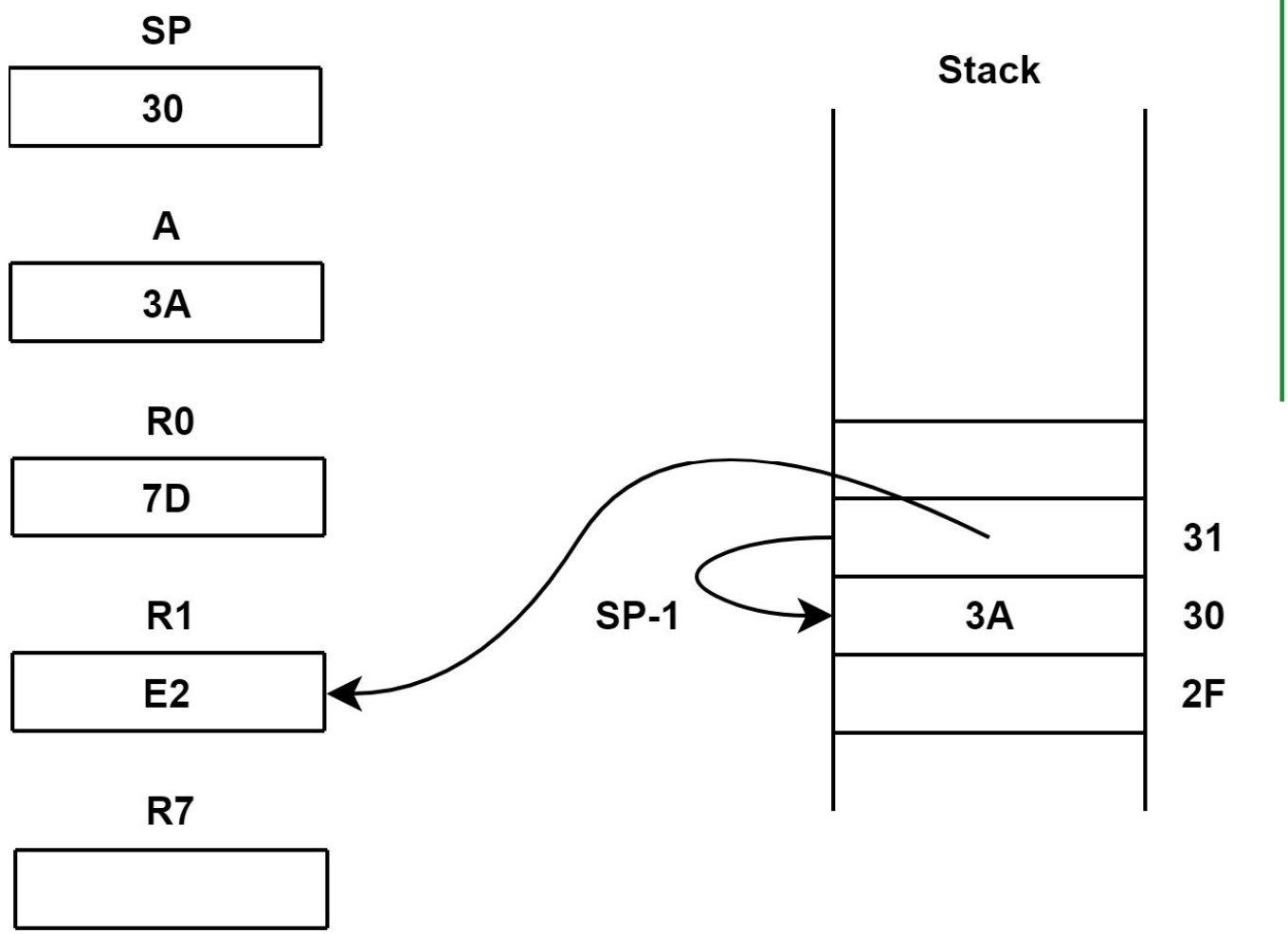


Stack – LIFO

Thực hiện lệnh

POP 1

lấy giá trị tại ngăn xếp đang được trỏ bởi SP đưa vào thanh ghi R1, SP= Sp-1



Stack – LIFO

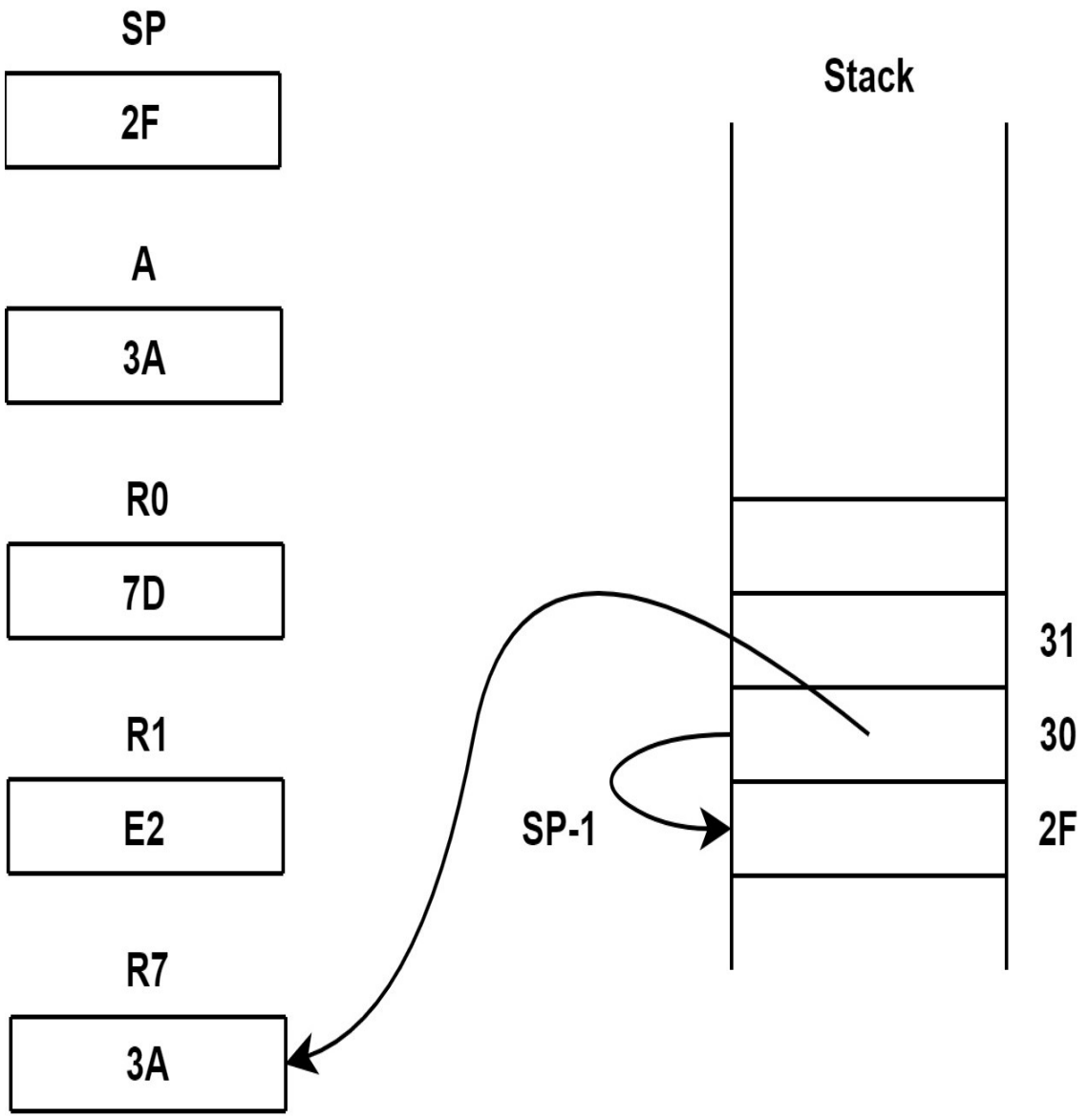
Thực hiện lệnh

POP 7

lấy giá trị tại ngăn xếp đang được trỏ bởi SP đưa vào thanh ghi R7, SP =SP - 1

Nhận xét

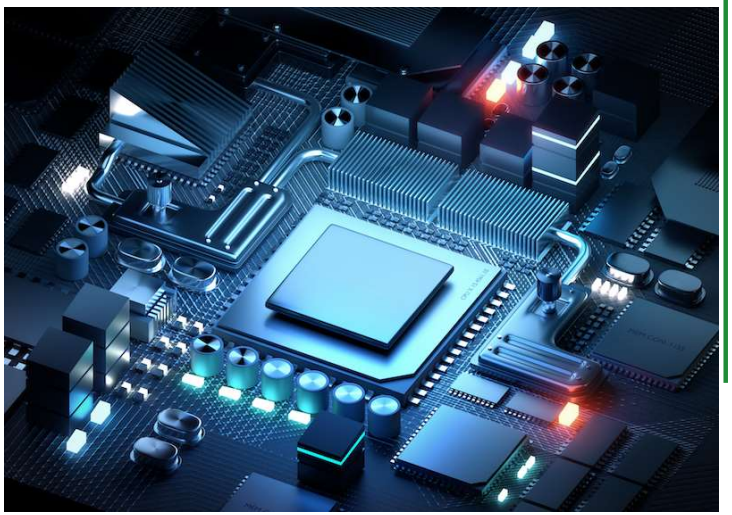
- Nếu số lệnh POP bằng số lệnh PUSH thì giá trị SP là không đổi
- Có thể cất vào 1 thanh ghi nhưng lấy ra vào 1 thanh ghi khác



Interrupt – Ngắt

Định nghĩa:

Ngắt là một sự kiện (có thể bất thường hoặc theo chu kì) được báo tới CPU, từ đó CPU có thể ra các quyết định có hay không thực hiện các chương trình phục vụ ngắt, điều này phụ thuộc mức độ ưu tiên của ngắt đó



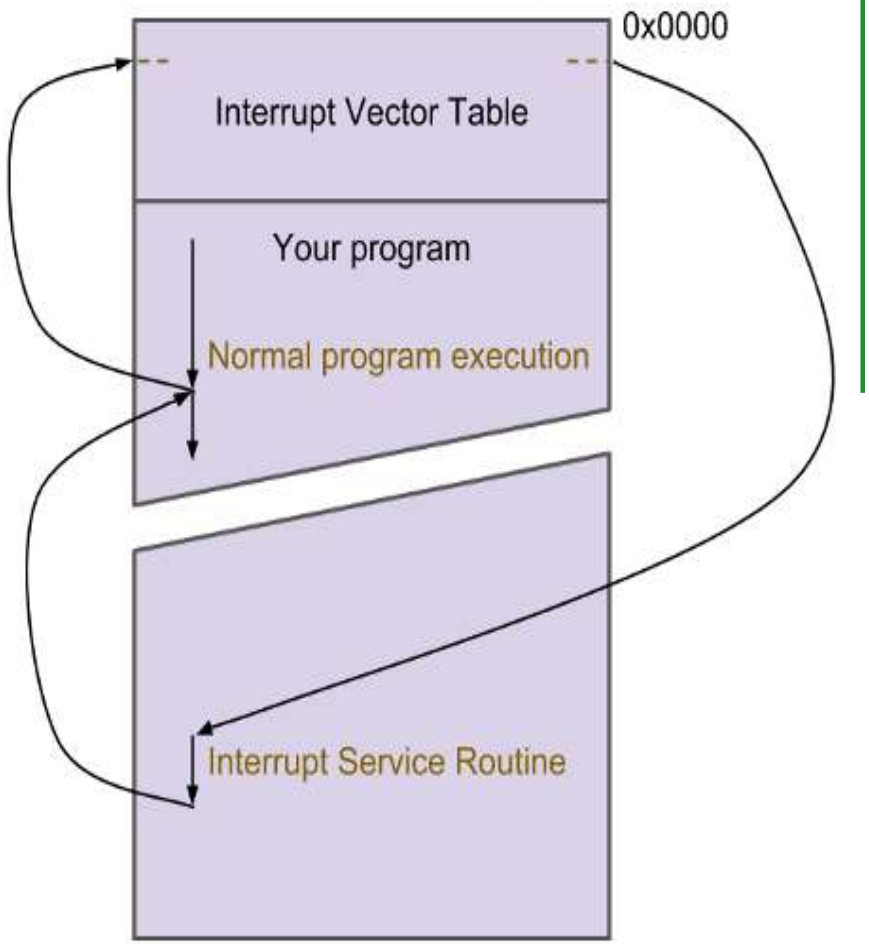
Phân loại:

- Ngắt theo sự kiện / ngắt theo thời gian (ngắt timer)
- Ngắt cứng (ngắt do phần cứng)/ ngắt mềm (ngắt do phần mềm)
- Ngắt che được (Mask Interrupt) / Ngắt không che được (Non Mask Interrupt)

Interrupt – Ngắt

Vector ngắt:

- Ngắt được mã hóa bởi vector ngắt, tập hợp các vector ngắt tạo thanh bảng vector ngắt
- Ngắt reset là ngắt cao nhất, là vector ngắt đầu tiên ở bảng vector ngắt
- Tùy thuộc CPU mà bảng vecor ngắt được đặt ở đầu hoặc cuối bộ nhớ



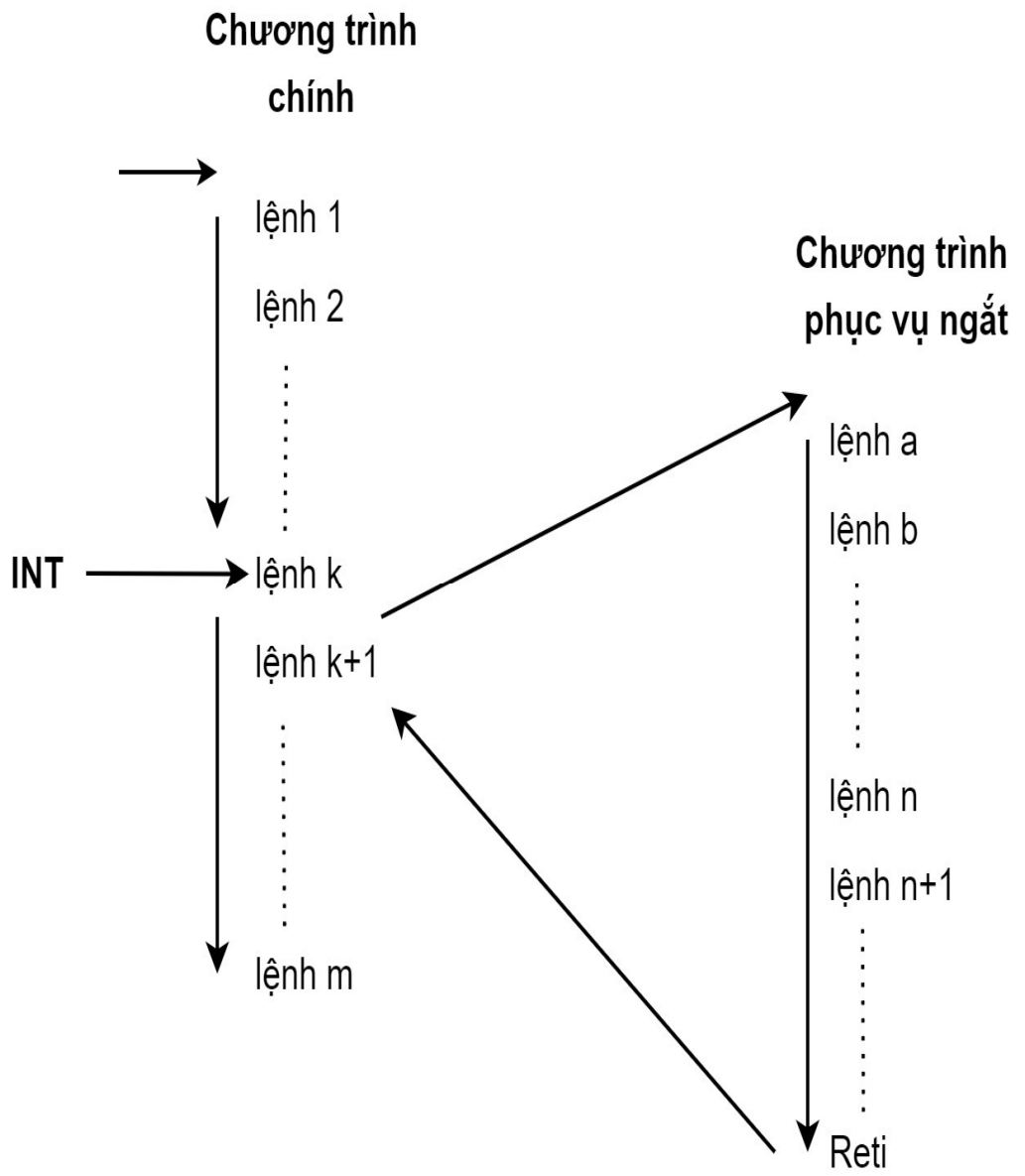
Interrupts	Memory Location
Reset	0000
Timer0	000B
Timer1	001B
INT0	0003
INT1	0013
Serial com	0023

Interrupt – Ngắt

Cách thực hiện ngắt

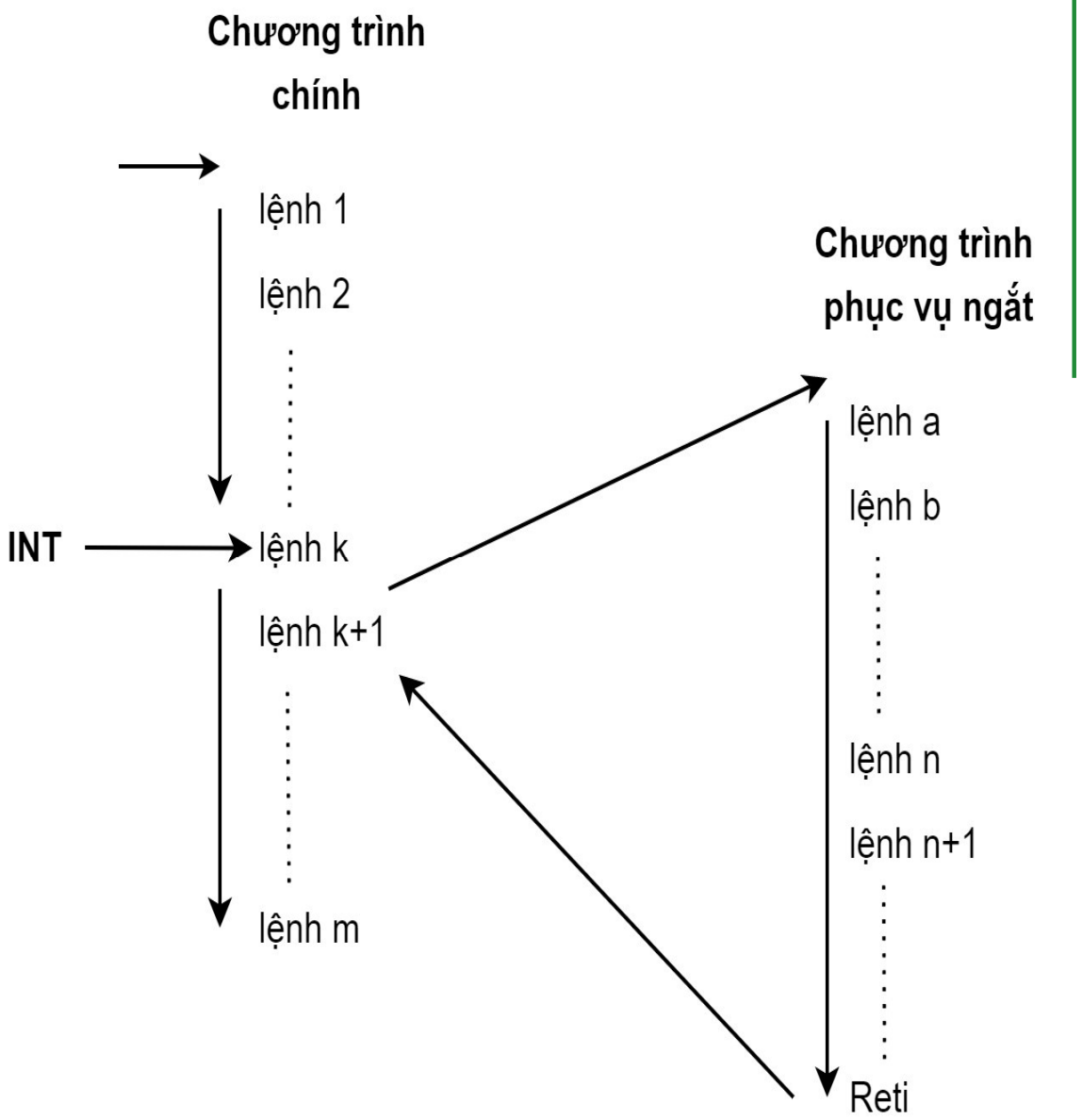
Chương trình, có ngắt tại lệnh k:

- Thực hiện xong lệnh k, sau khi thực hiện xong thì địa chỉ của lệnh k+1 được nạp vào PC
- CPU kiểm tra xem là loại ngắt gì (kiểm tra vector ngắt)
- Nếu là ngắt không che được sẽ cất giá trị PC vào trong ngăn xếp. Tiếp theo, CPU cất giá trị của các thanh ghi liên quan rồi đến các cờ

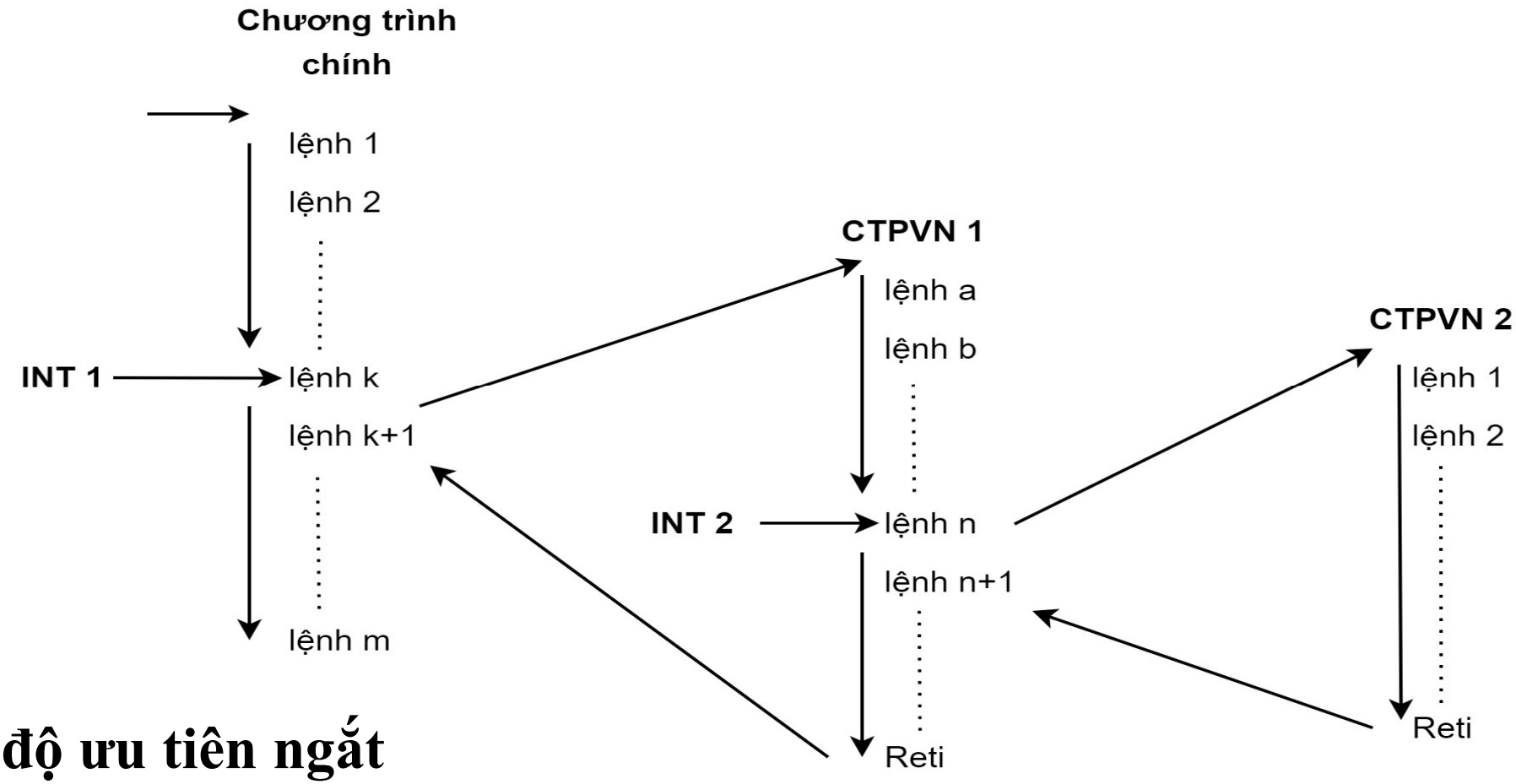


Interrupt – Ngắt

- PC được nạp vào địa chỉ của lệnh đầu tiên của chương trình phục vụ ngắt. Địa chỉ của lệnh a là địa chỉ đầu tiên của vector ngắt
- Kết thúc chương trình ngắt là lệnh **Reti** dùng để quay lại chương trình trước khi có ngắt, khi quay về, CPU khôi phục lại các cờ, tiếp đến là các thanh ghi liên quan và cuối cùng là giá trị PC (chứa địa chỉ của lệnh k+1)



Interrupt – Ngắt



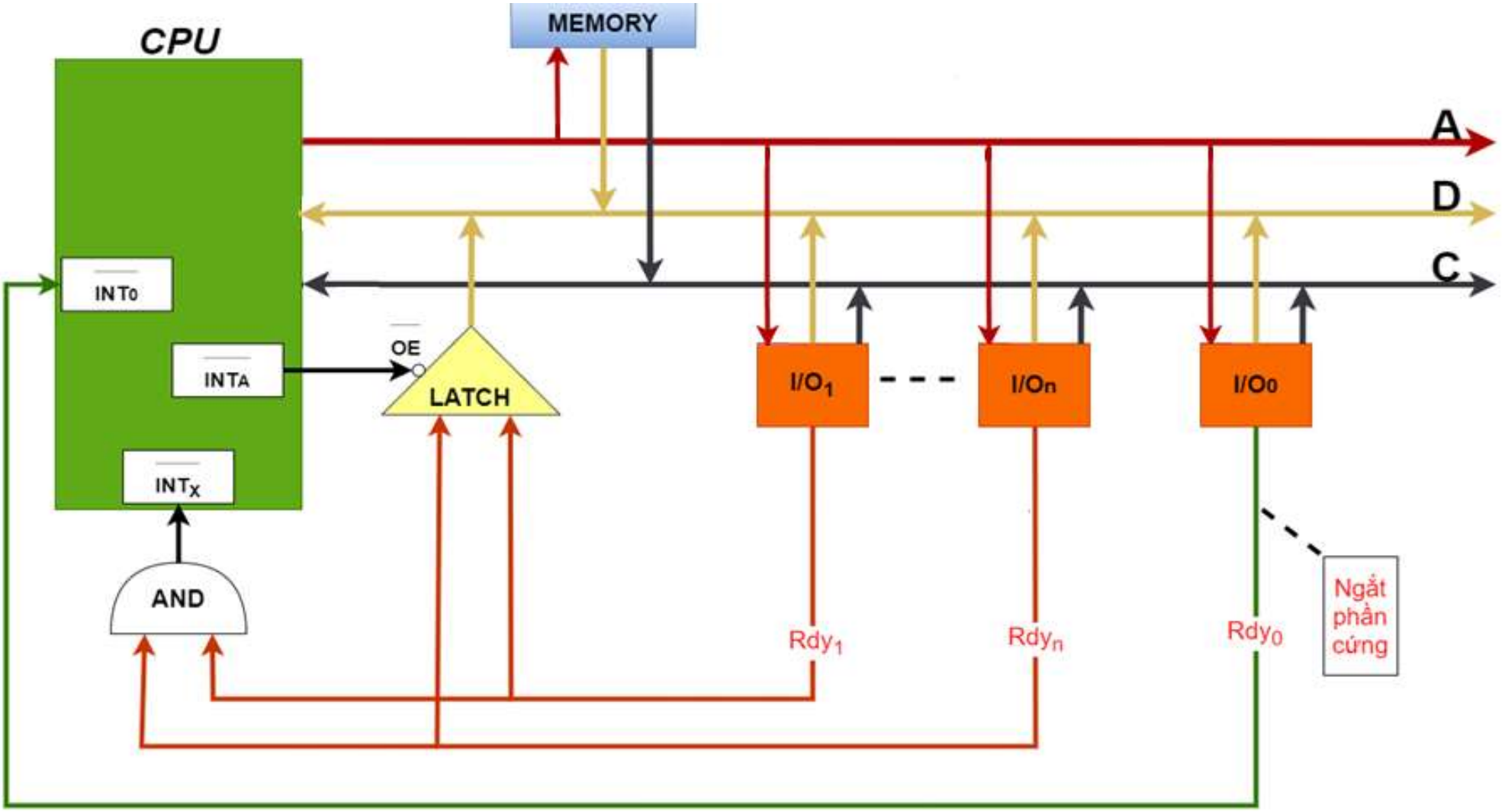
Mức độ ưu tiên ngắt

Nếu đang CTPVN1 lại có ngắt tiếp thì sẽ kiểm tra xem ưu tiên của hai ngắt để CPU thực hiện Int 1 hay Int 2

Để phân biệt (định nghĩa) ưu tiên ngắt sử dụng: thanh ghi IP (interrupt Piority)

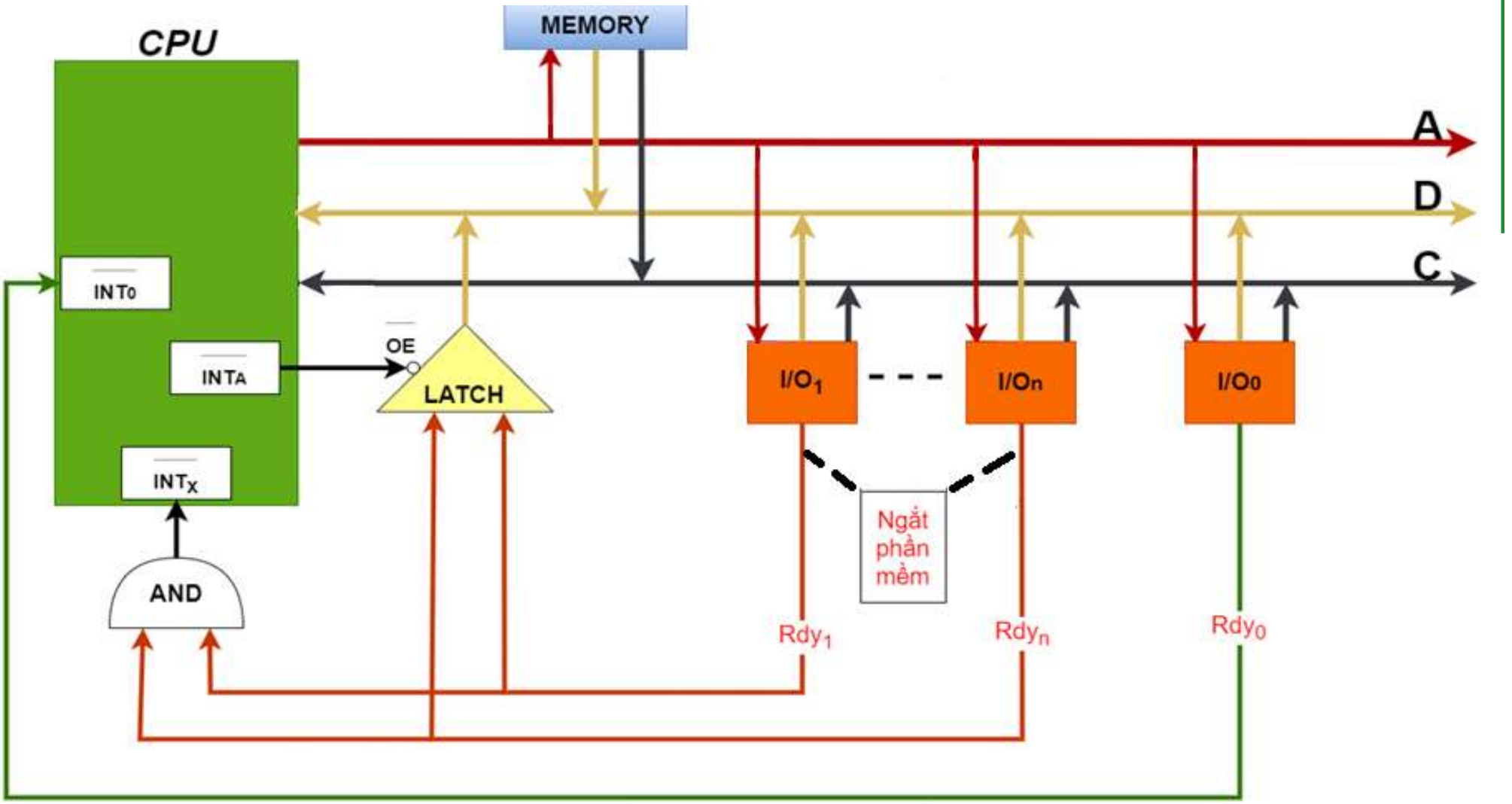
Interrupt – Ngắt

- Cách xác định nguồn báo ngắt bằng phần cứng



Interrupt – Ngắt

- Cách xác định nguồn báo ngắt bằng phần mềm



Interrupt – Ngắt

- Khi có tín hiệu ngắt => $\overline{INTx} = 0$ và CPU sẽ trả lời tín hiệu xác nhận ngắt $\overline{INTA}$, đưa tín hiệu vào $\overline{OE}$ mở latch đưa dữ liệu lên databus
- Ngay sau đó 1 chu kỳ máy, CPU đọc dữ liệu từ databus, xác định đó là ngoại vi số **x** và tra bảng vector ngắt để thực hiện chương trình phục vụ ngắt tương ứng.
- Khi có nhiều nguồn báo ngắt:

	Rdy1	Rdy2	Rdy3	Rdy4	Rdy5	Rdy6	Rdy7	Rdy8
Ban đầu : $\overline{INTx} = 1$	1	1	1	1	1	1	1	1
Rdy2=Rdy5=0	1	**0**	1	1	1	1	1	**0**

- Sau khi CPU đọc dữ liệu từ databus vào thanh ghi, và kiểm tra giá trị Reg và mức độ ưu tiên ngắt để quyết định thực hiện CTpvn tương ứng

Ngắt- MCS 51

- Đối với MCS51, người dùng có thể sử dụng 5 véc-tơ ngắt, tuy nhiên, trên thực tế, MCS51 có 6 nguồn báo ngắt (bao gồm cả ngắt RESET- luôn ở đầu tiên của bảng vector ngắt)

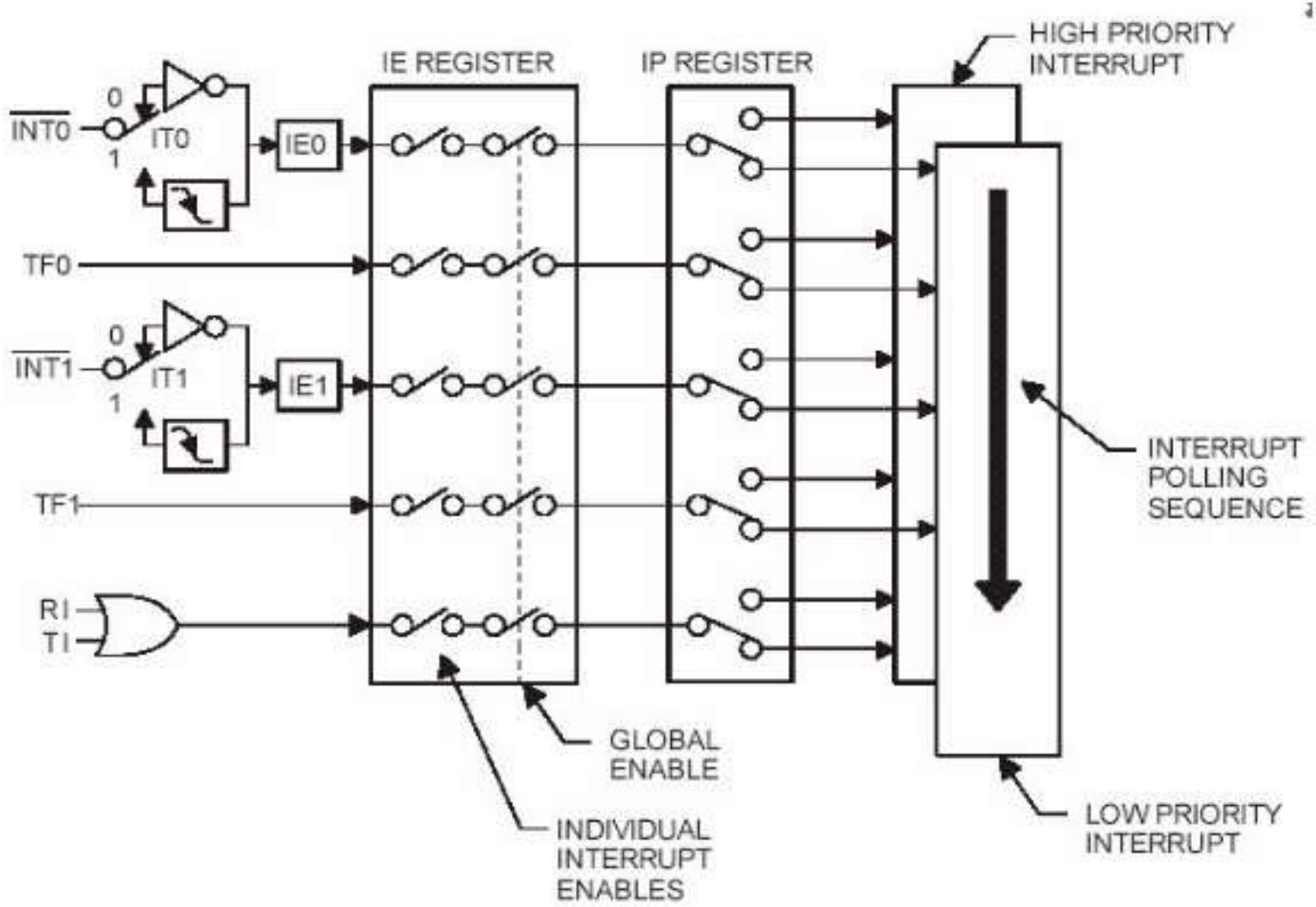
Ngắt ngoài 0

Ngắt Timer 0

Ngắt ngoài 1

Ngắt Timer 1

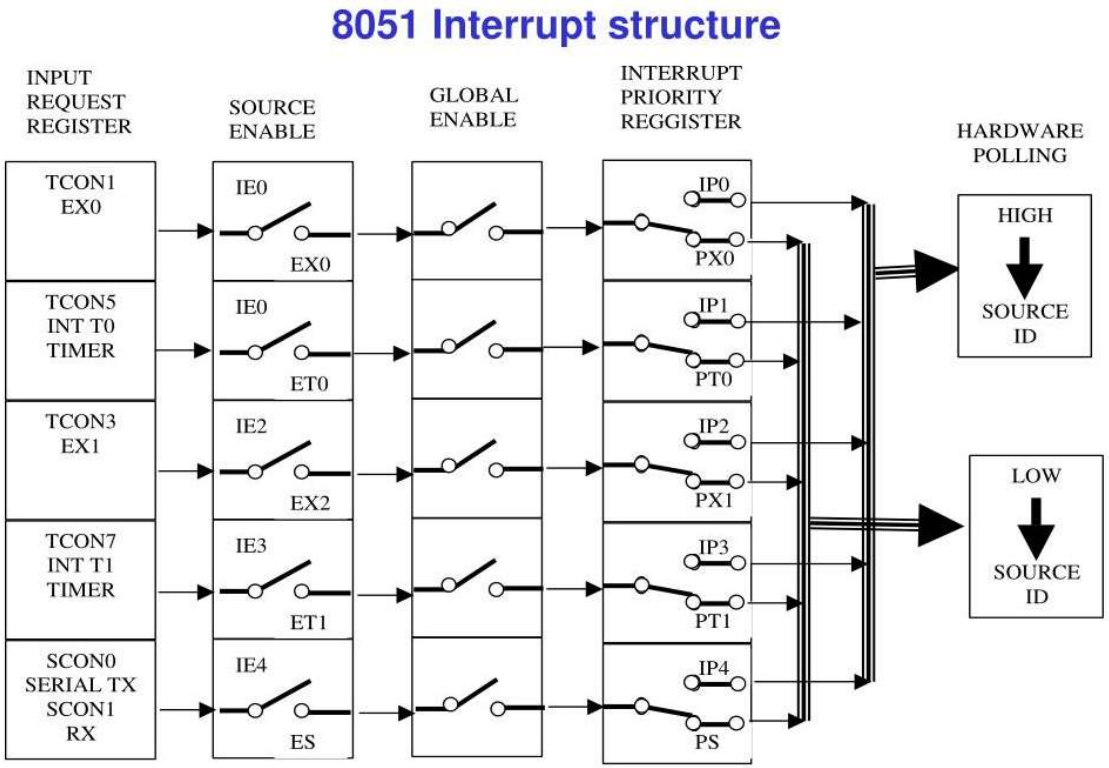
Ngắt nối tiếp



Ngắt- MCS 51

- Bảng vector ngắt
- Sử dụng ngắt trong MCS-51
  - Setb EA (enable all) (IE register)
  - Setb Ngắt tương ứng (IE register)
  - CT phục vụ ngắt tại địa chỉ vector ngắt

Ngắt	Kí hiệu	Int Vector Addr
RESET	RESET	0000H
Ngắt ngoài 0	IE0	0003H
Timer 0	TF0	000BH
Ngắt ngoài 1	IE1	0013H
Timer 1	TF1	001BH
Ngắt nối tiếp	RI&TI	0023H



- Mức độ ưu tiên ngắt : IP

Ngắt- MCS 51

- IE – thanh ghi kích hoạt ngắt, cho phép truy cập theo từng bit.

EA	-	ET2	ES	ET1	EX1	ET0	EX0

EA	IE.7	Cho phép/cấm hoạt động của cả thanh ghi
-	IE.6	Chưa sử dụng
ET2	IE.5	Cho phép ngắt timer 2 (chỉ với 8052)
ES	IE.4	Cho phép ngắt cổng truyền thông nối tiếp
ET1	IE.3	Cho phép ngắt timer 1
EX1	IE.2	Cho phép ngắt ngoài 1
ET0	IE.1	Cho phép ngắt timer 0
EX0	IE.0	Cho phép ngắt ngoài 0

Ngắt- MCS 51

- Thiết lập cho ngắt ngoài 1, timer 0 và truyền tin.

7	6	5	4	3	2	1	0
1	0	0	1	0	1	1	0

Lệnh: mov IE, #10010110B

Cách viết khác:	setb	ET0	hoặc	setb	IE.1
	setb	EX1		setb	IE.2
	setb	ES		setb	IE.4
	setb	EA		setb	IE.7

Ngắt- MCS 51

- IP – thanh ghi định nghĩa cấp độ ưu tiên ngắt

-	-	PT2	PS	PT1	PX1	PT0	PX0

-	IP.7	Chưa sử dụng
-	IP.6	Chưa sử dụng
PT2	IP.5	Định nghĩa mức ưu tiên cho ngắt timer 2 (chỉ với 8052)
PS	IP.4	Định nghĩa mức ưu tiên cho ngắt cổng truyền thông nối tiếp
PT1	IP.3	Định nghĩa mức ưu tiên cho ngắt timer 1
PX1	IP.2	Định nghĩa mức ưu tiên cho ngắt ngoài 1
PT0	IP.1	Định nghĩa mức ưu tiên cho ngắt timer 0
PX0	IP.0	Định nghĩa mức ưu tiên cho ngắt ngoài 0

Ngắt- MCS 51

- Mặc định thứ tự ưu tiên lần lượt cho từng ngắt là: IE0, TF0, IE1, TF1, RI&TI.

7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0

Thứ tự ưu tiên: TF0 -> RI/TI -> IE0 -> IE1 -> TF1

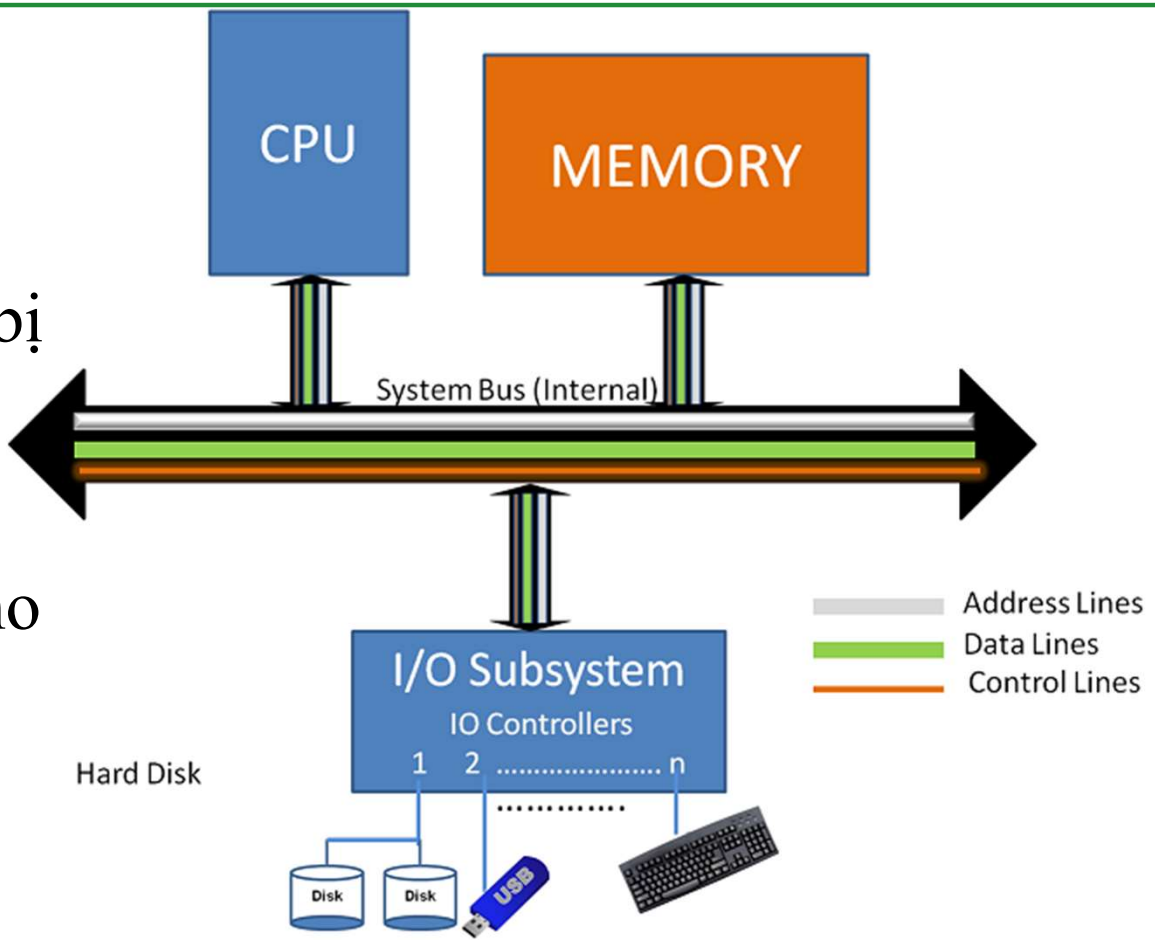
Lệnh: mov IP, #00010010B

Cách viết khác: setb PT0 hoặc setb IP1

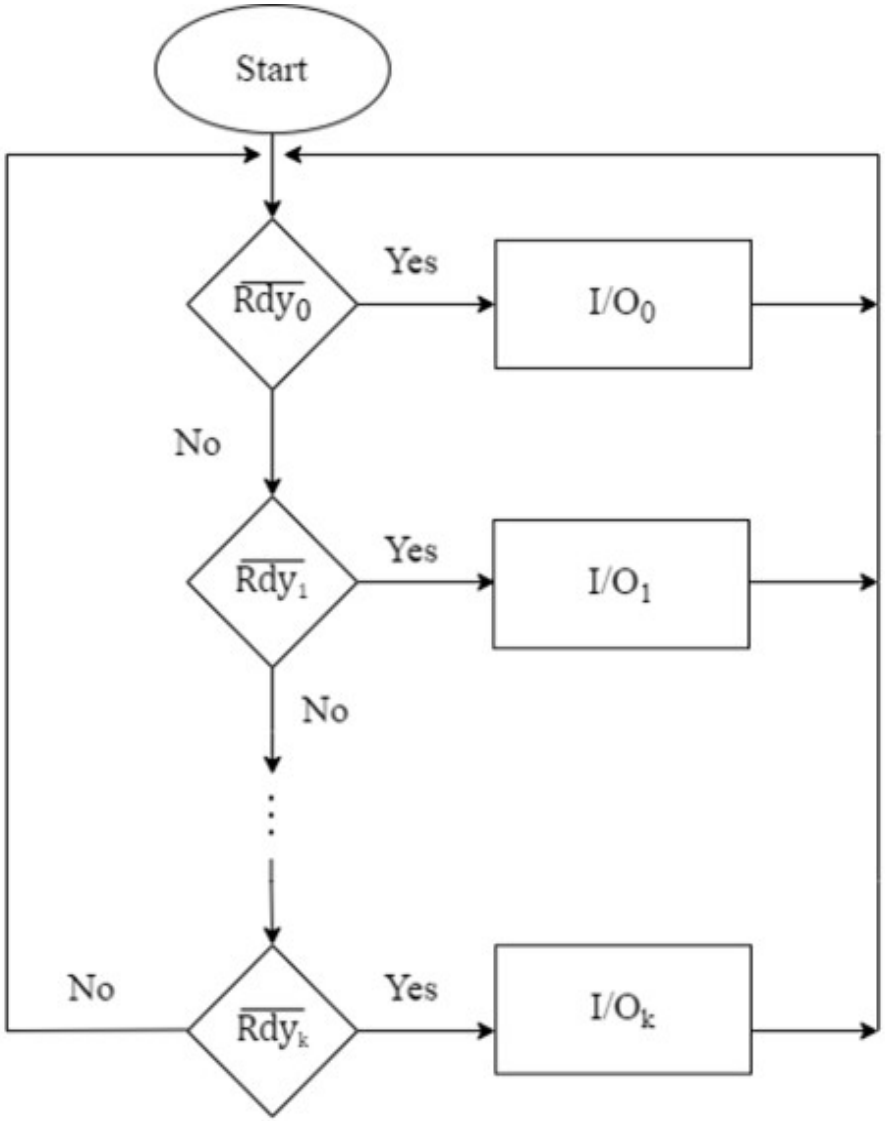
setb PS setb IP4

Các phương pháp vào ra – Vào ra bằng chương trình

- Sử dụng một chương trình để điều khiển việc trao đổi dữ liệu giữa CPU và thiết bị ngoại vi.
- Điều kiện: Mỗi 1 ngoại vi cần phải có tín hiệu báo cho CPU biết về việc sẵn sàng trao đổi dữ liệu $\overline{Rdy_x}$

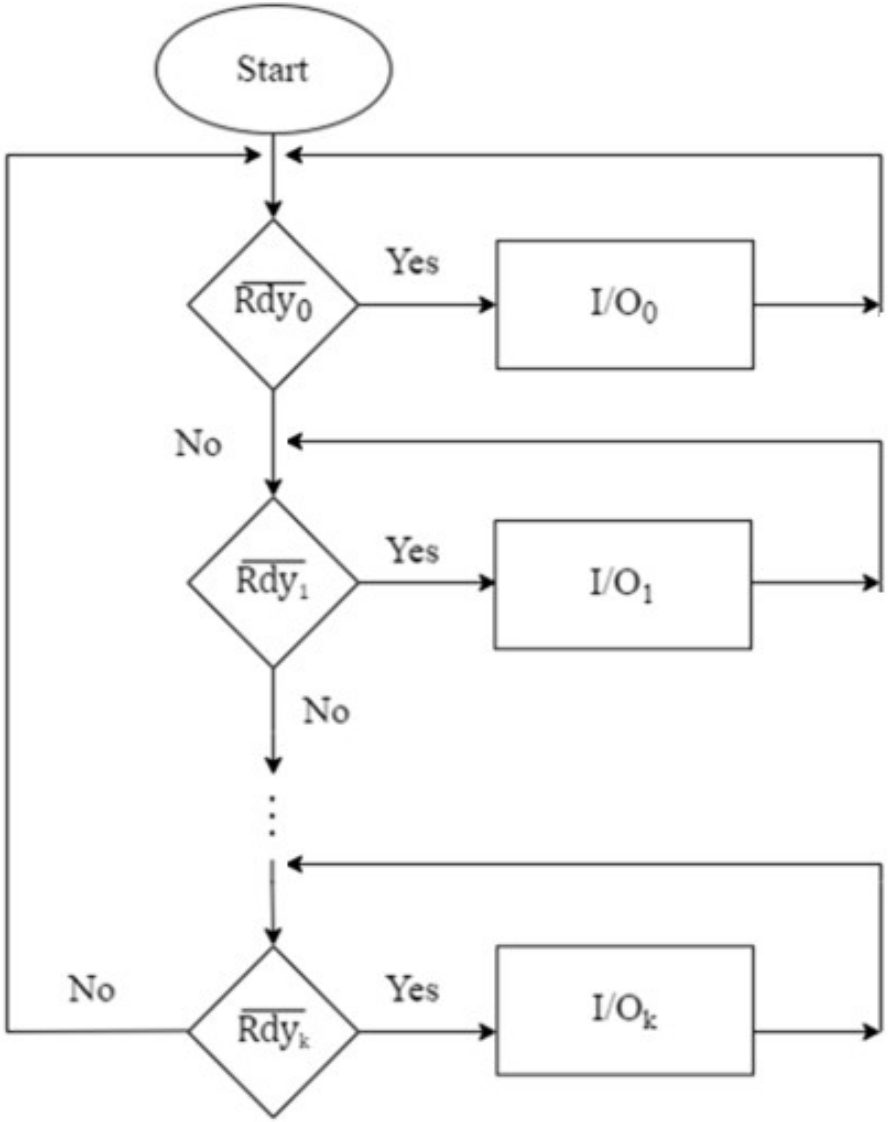


Các phương pháp vào ra – Vào ra bằng chương trình



- Phương pháp ưu tiên tuyệt đối

- Phương pháp ưu tiên quay vòng



Các phương pháp vào ra – Vào ra bằng chương trình

- Ưu điểm :
  - Dễ viết chương trình .
  - Người lập trình có thể thay đổi mức độ ưu tiên.
- Nhược điểm:
  - Tốn thời gian để hỏi từng ngoại vi dù không có nhu cầu.
  - Khi 1 ngoại vi thứ k bị hỏng -> Làm cho ngoại vi thứ k + 1, k + 2 không được kiểm tra.

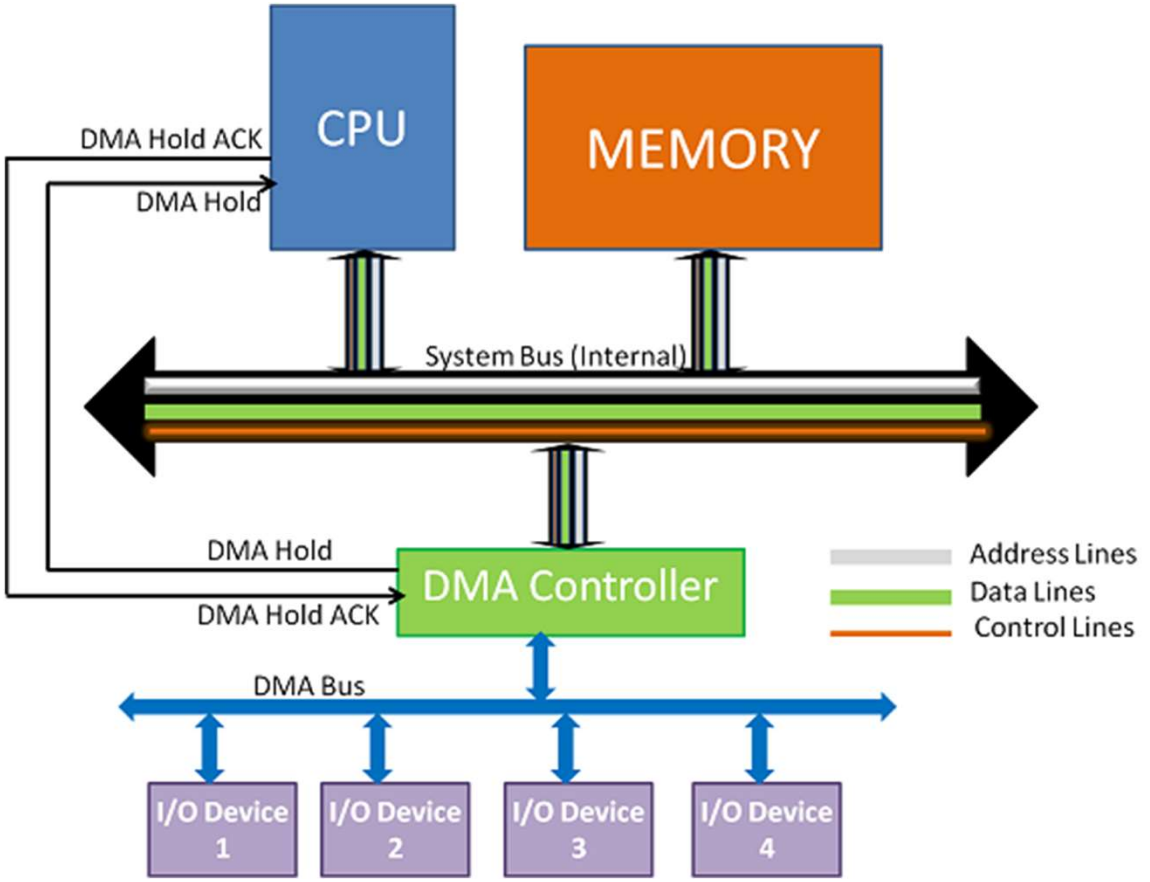
Các phương pháp vào ra – Vào ra bằng ngắt

- Xem lại phần Ngắt trong hệ Vi xử lý :
- Ưu điểm: CPU không tốn thời gian hỏi các I/O không có nhu cầu trao đổi dữ liệu.
- Nhược điểm : Khi ngắt xảy ra mà lượng trao đổi dữ liệu lớn, CPU sẽ tốn nhiều thời gian với ngắt đó.

Các phương pháp vào ra – Vào ra DMA

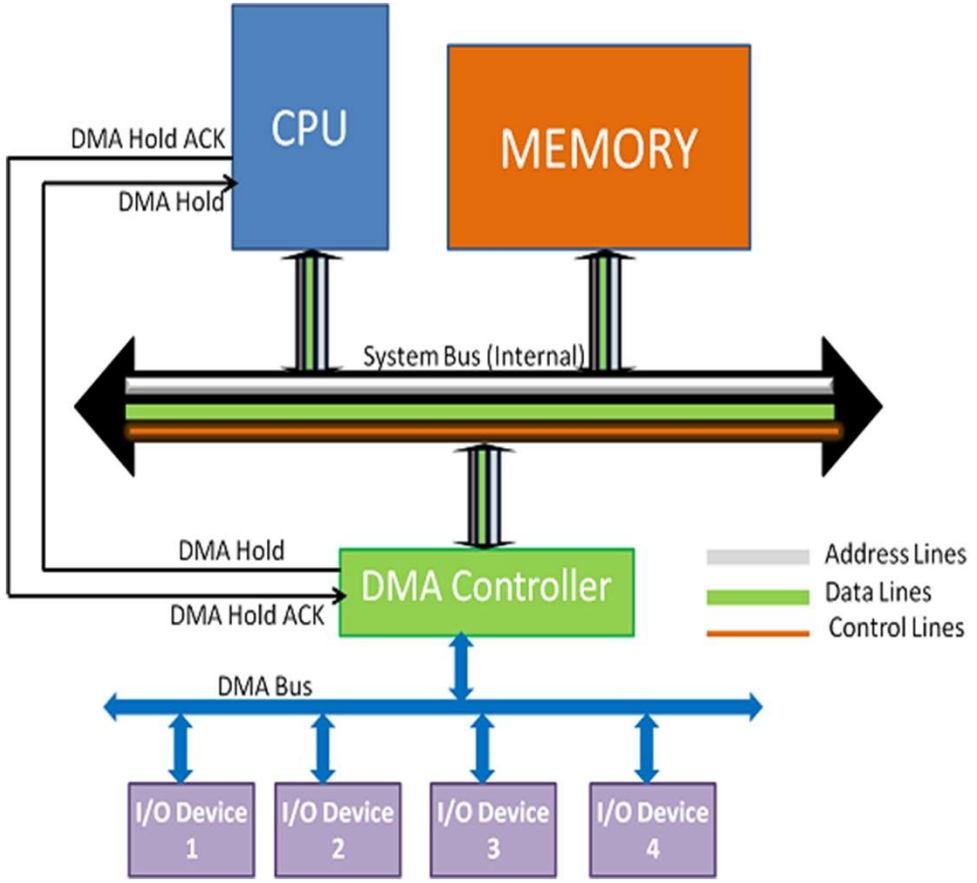
- DMA – Direct Memory Access: Phương pháp ngắt, việc trao đổi dữ liệu giữa I/O và Mem cần phải thông qua CPU bằng databus.

- DMA cho phép việc trao đổi dữ liệu giữa I/O và Mem trực tiếp, không cần thông qua CPU.
- DMAC (DMA Controller) thay CPU điều khiển quá trình trao đổi dữ liệu I/O và Mem.
- DMA cải thiện tốc độ trao đổi dữ liệu và lượng dữ liệu lớn.



Các phương pháp vào ra – Vào ra DMA

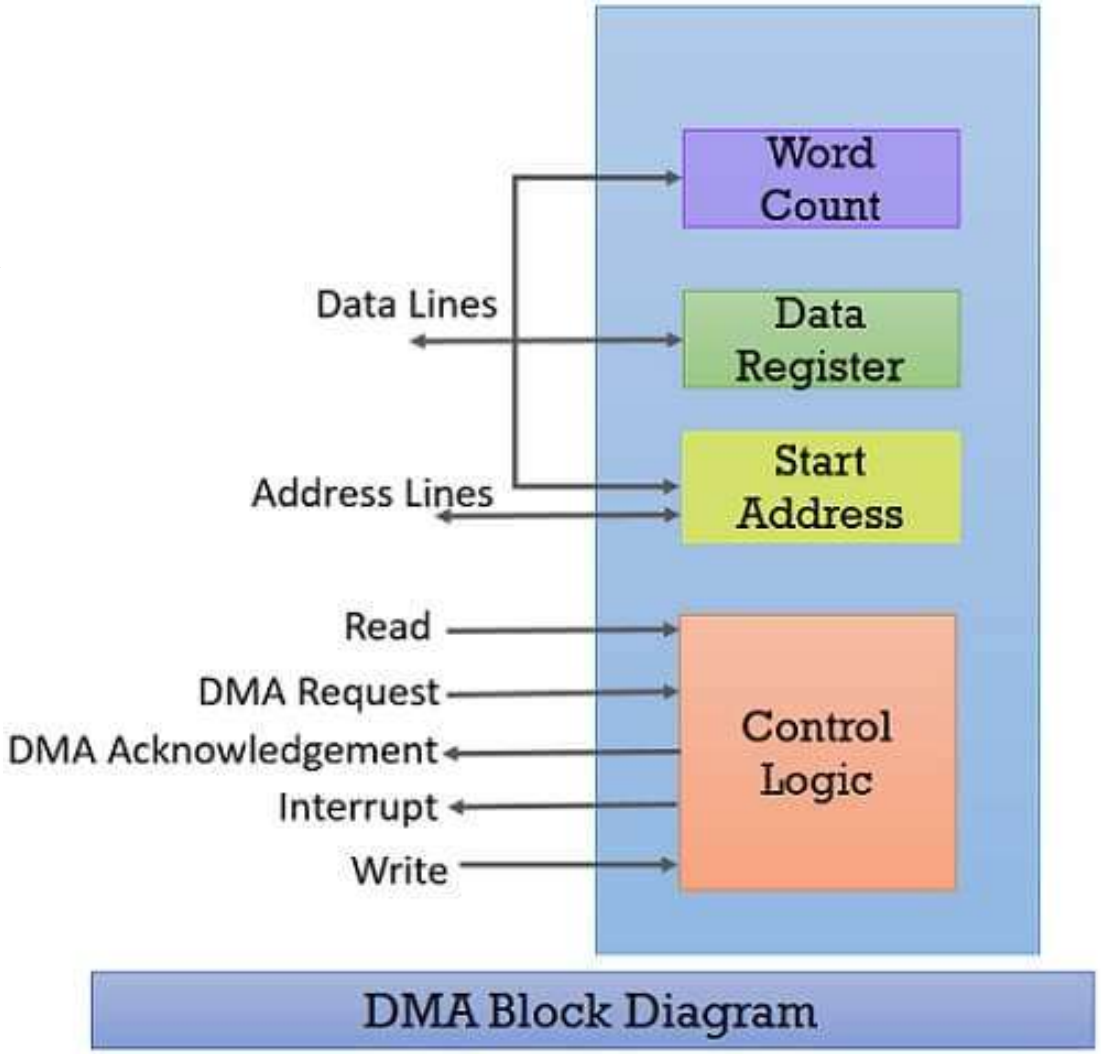
- I/O gửi ***DMA Req*** tới DMAC
- DMAC gửi ***DMA Hold*** tới CPU
- CPU sẽ kiểm tra các điều kiện cần thiết trước khi cho phép DMA
- Đủ đk: CPU trả lời ***DMA Hold Ack*** đến DMAC. Đồng thời, CPU đưa tất cả các đường bus lên trạng thái cao trở và CPU tách hoàn toàn ra khỏi hệ thống) nhường quyền điều khiển cho DMAC



- DMAC gửi tín hiệu xác nhận ***DMA Ack*** tới I/O cho phép trao đổi dữ liệu I/O và Mem.

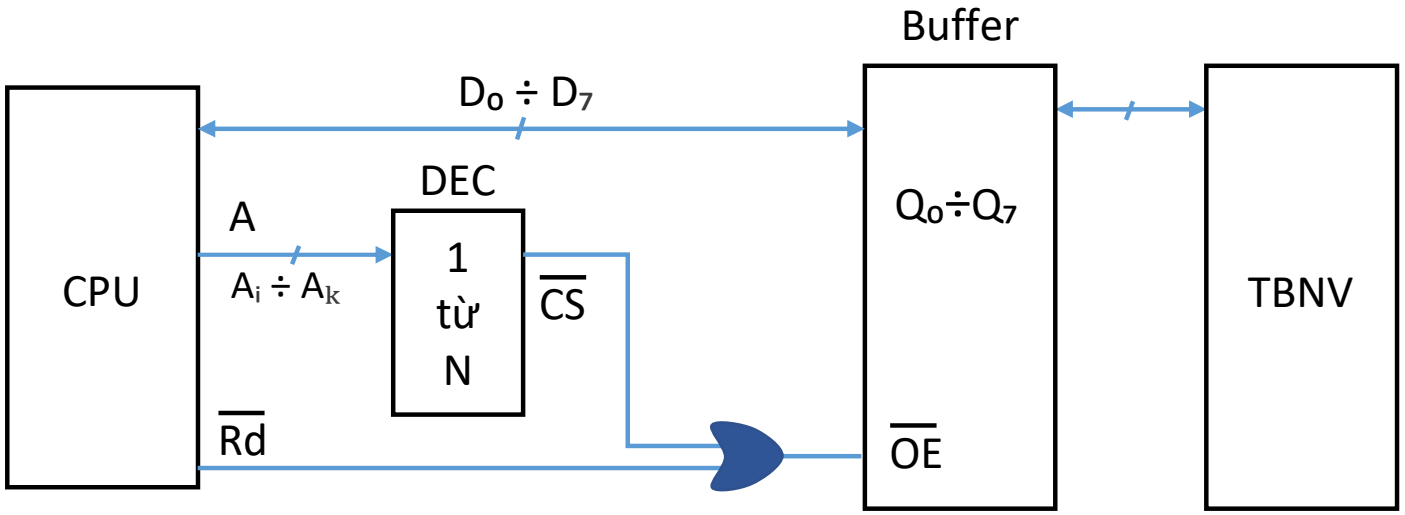
Các phương pháp vào ra – Vào ra DMA

- DMAC xác định từ địa chỉ đầu, vùng nhớ cần trao đổi, tiếp đó I/O thực hiện trao đổi dữ liệu với Mem.
- Khi kết thúc trao đổi dữ liệu, I/O gửi tín hiệu tới DMAC, DMAC gửi tín hiệu về CPU. CPU lấy lại quyền điều khiển đường bus, phản hồi về DMAC và ngoại vi, kết thúc quá trình.
- Nhược điểm DMA và hướng khắc phục

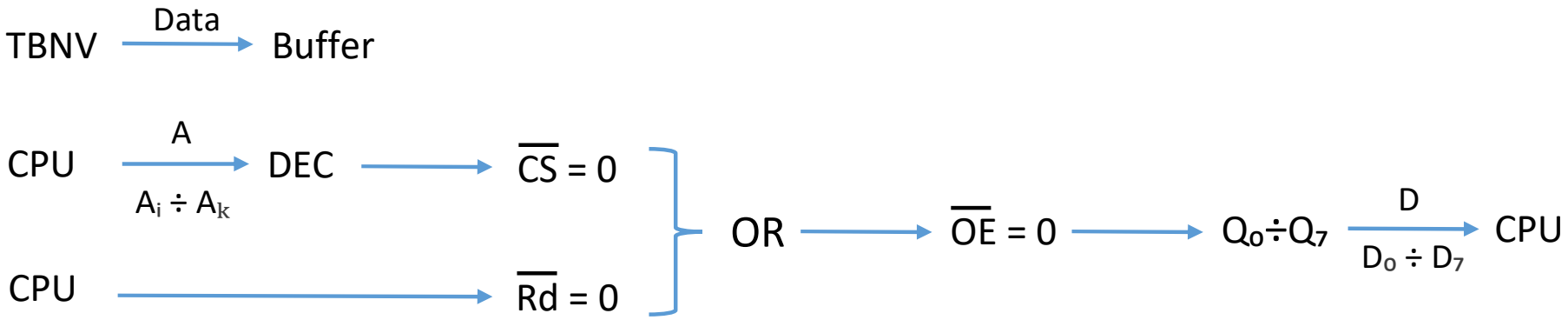


Cổng vào ra

- Cổng vào đơn giản:
  - Mục tiêu: Đọc dữ liệu từ thiết bị ngoại vi, đưa vào CPU

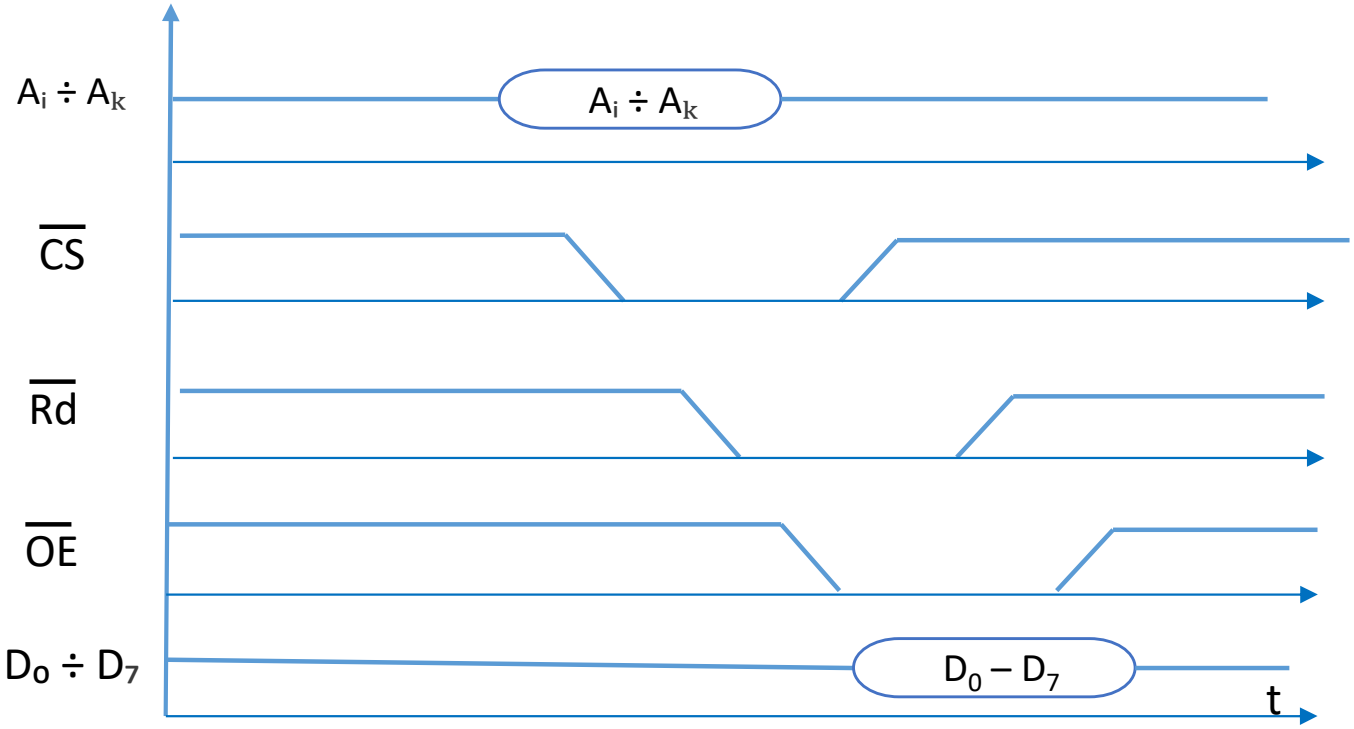


  - Nguyên tắc hoạt động:



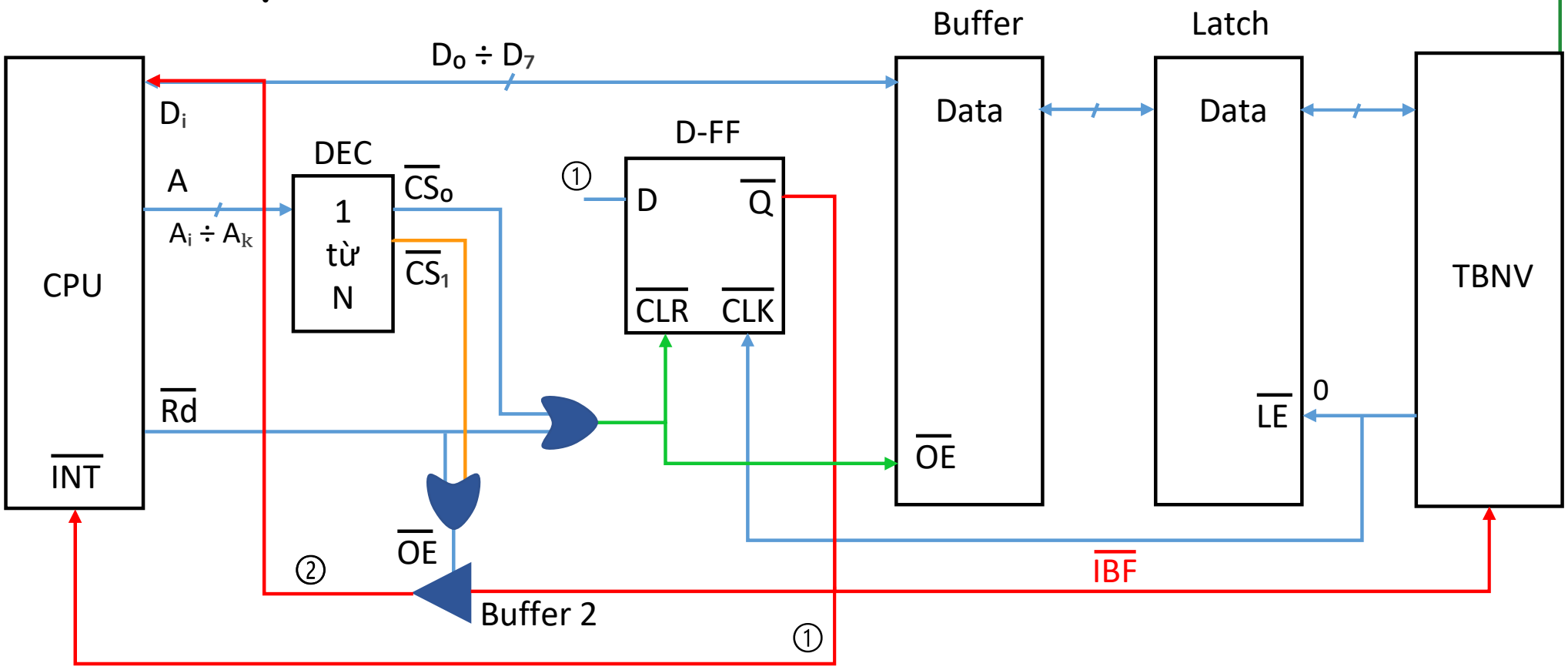
Cổng vào ra

- Giản đồ thời gian:



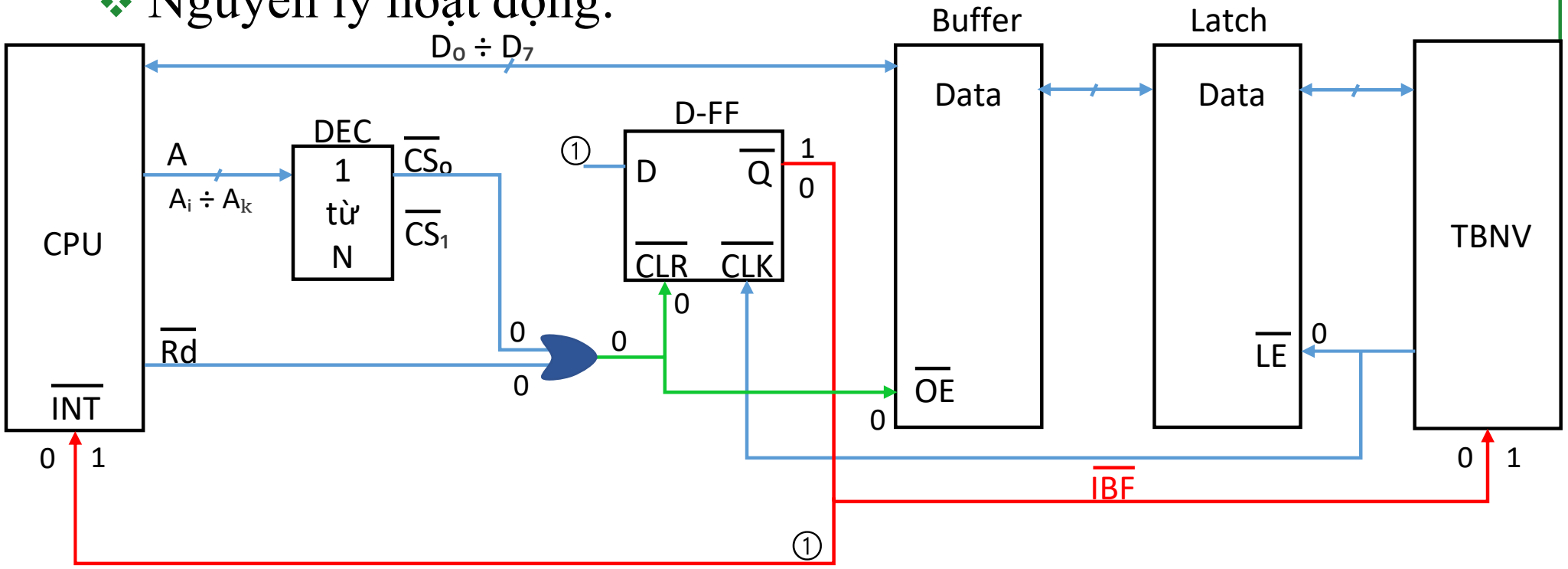
Cổng vào ra

- Cổng vào có đối thoại:
  - Cấu tạo:



Cổng vào ra

- Cổng vào có đối thoại:
  - Nguyên lý hoạt động:

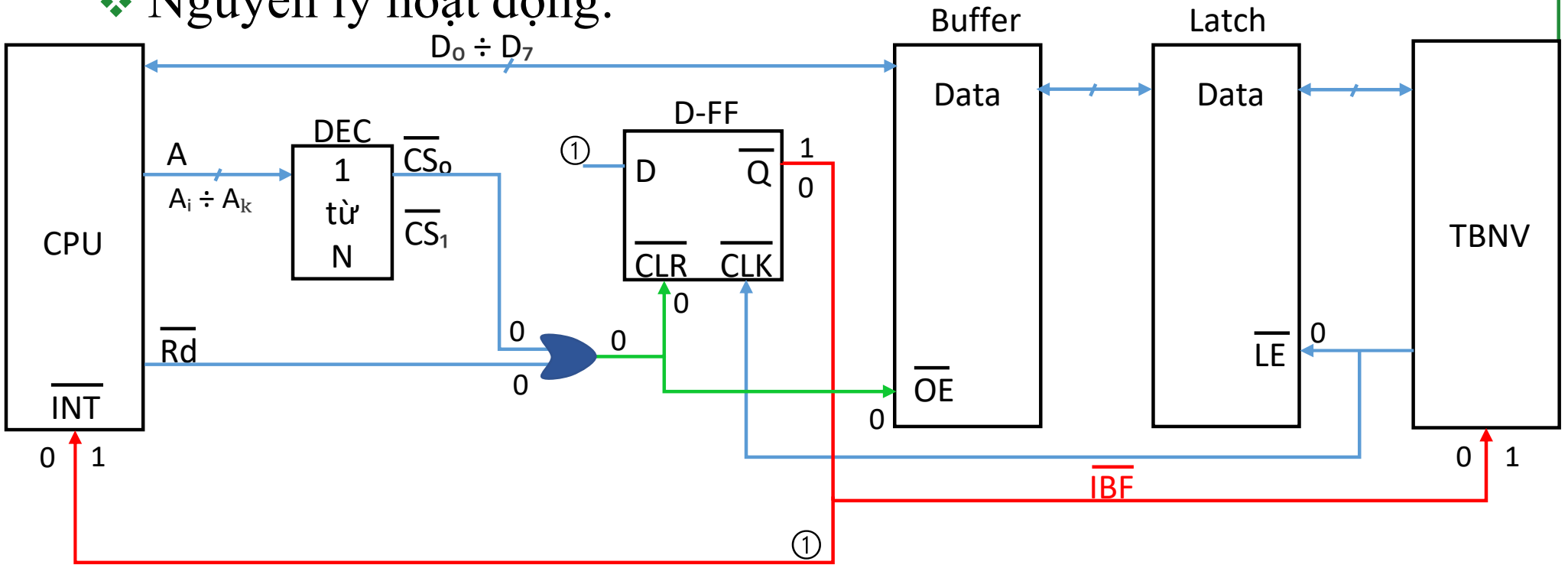


TBNV → Data → Latch TBNV → $\overline{LE} = 0$ → Latch → Data → Buffer

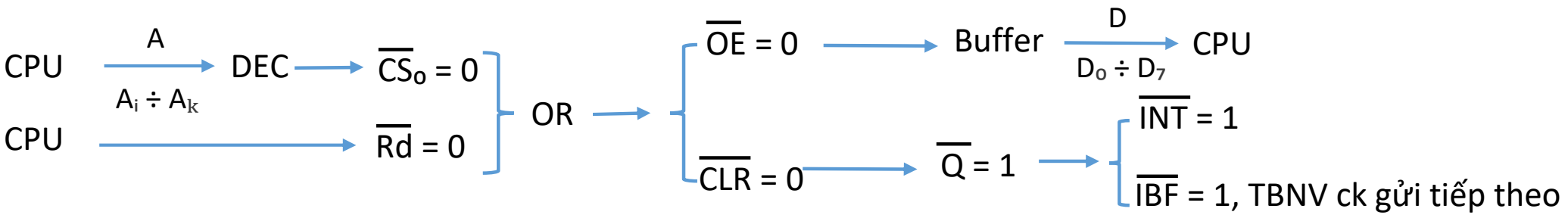
TBNV → $\overline{CLK}$ → D_FF = 1 → Q = D = 1 → $\overline{Q} = 0$ D-FF → ($\overline{IBF} = 0$, Input Buffer Full) → TBNV và Int CPU

Cổng vào ra

- Cổng vào có đối thoại:
  - Nguyên lý hoạt động:

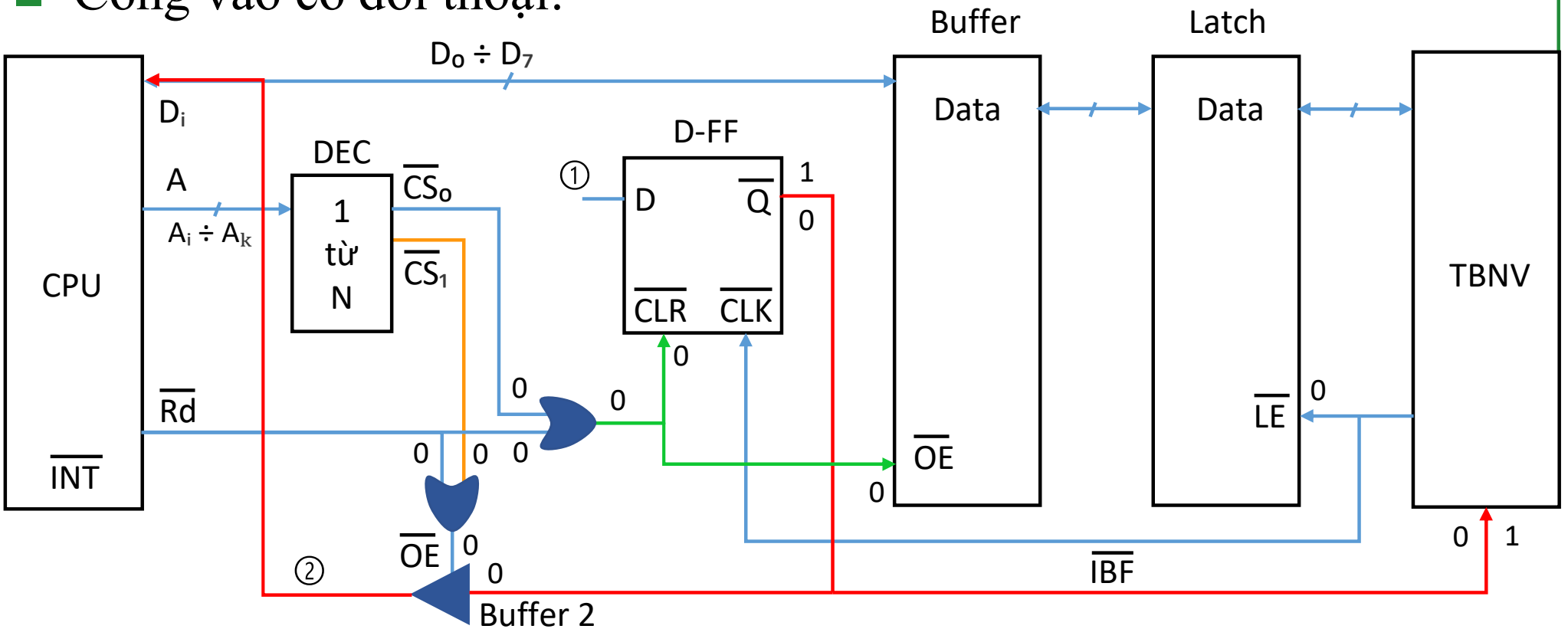


Cách 1: Đi vào chân ngắt của CPU, CPU ngắt



Cổng vào ra

- Cổng vào có đối thoại:



Cách 2: CPU đọc khi muốn

$D_i = 0$ → TBNV cần trao đổi dữ liệu

CPU tiến hành các bước để đọc dữ liệu:

CPU $\xrightarrow[A_i \div A_k]{A}$ DEC → $\overline{CS_1} = 0$

CPU → $\overline{Rd} = 0$

$\overline{OE} = 0$, $\overline{Q} = 0$ → Buffer 2 $\xrightarrow[D_i]{②}$ CPU

CPU $\xrightarrow[A_i \div A_k]{A}$ DEC → $\overline{CS_o} = 0$

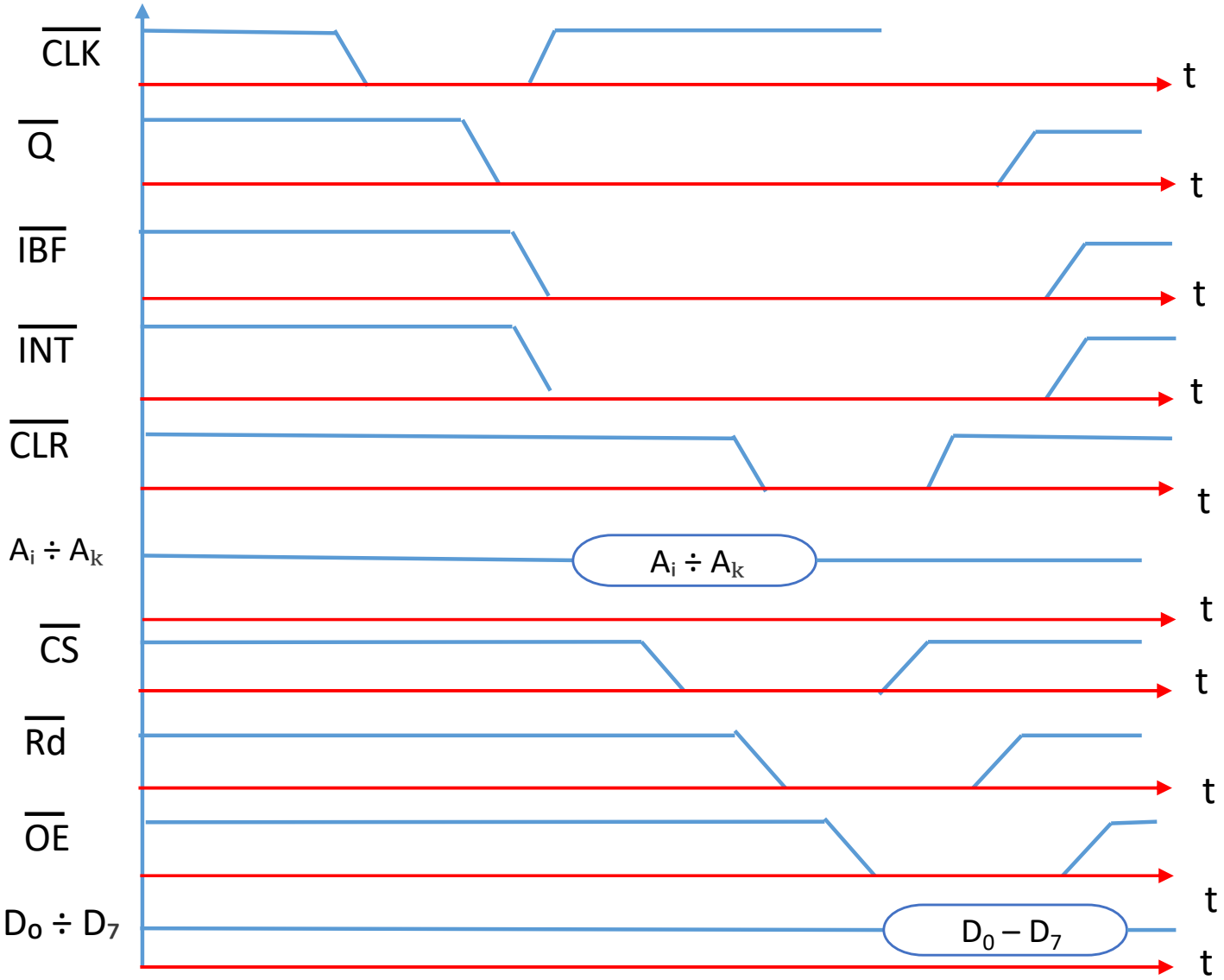
CPU → $\overline{Rd} = 0$

OR →

- $\overline{OE} = 0$ → Buffer $\xrightarrow[D_o \div D_7]{D}$ CPU
- $\overline{CLR} = 0$ → $\overline{Q} = 1$ → $\overline{IBF} = 1$ → TBNV ck tiếp

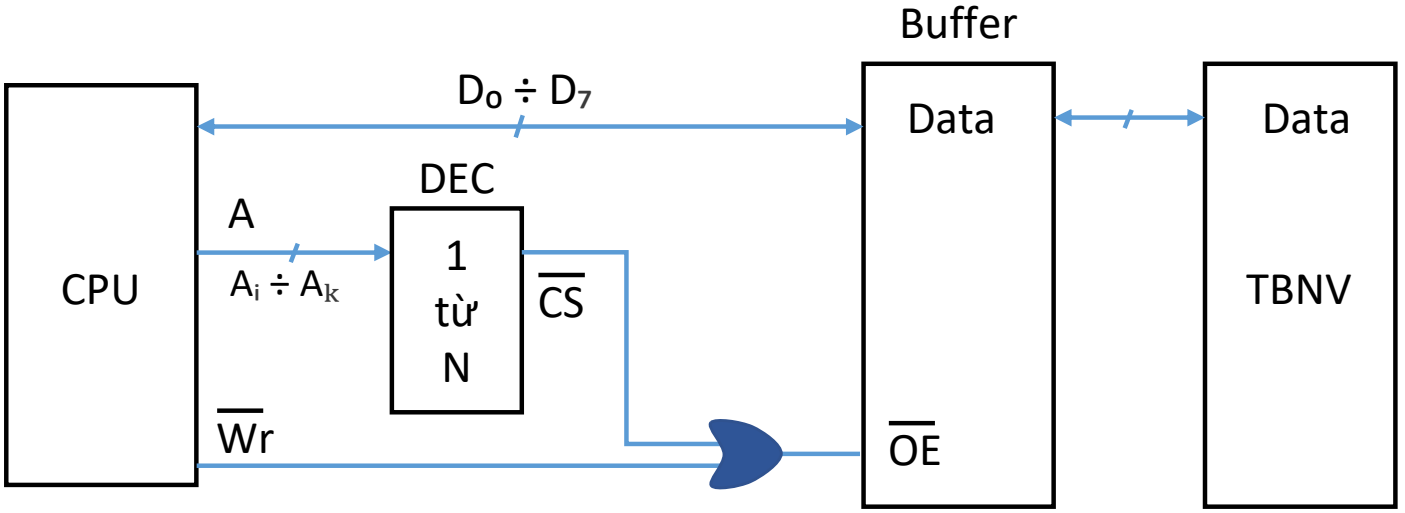
Cổng vào ra

- Cổng vào có đối thoại:

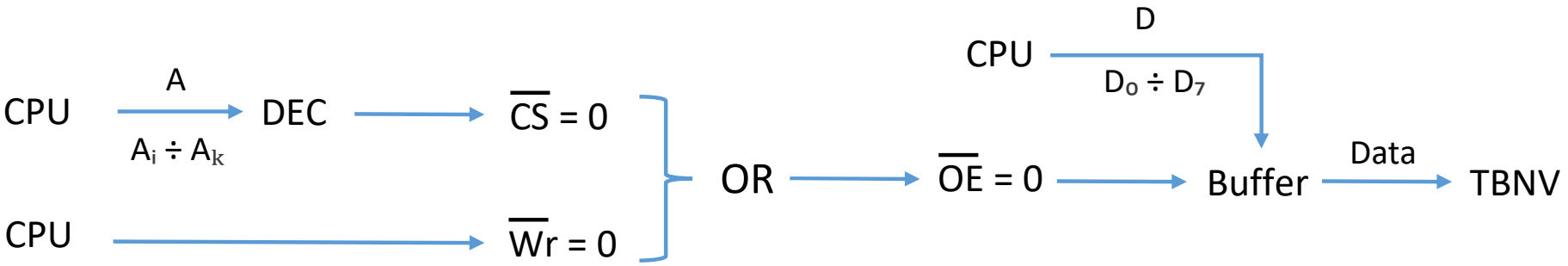


Cổng vào ra

- Cổng ra đơn giản:
  - Mục tiêu: Viết dữ liệu từ CPU ra thiết bị ngoại vi

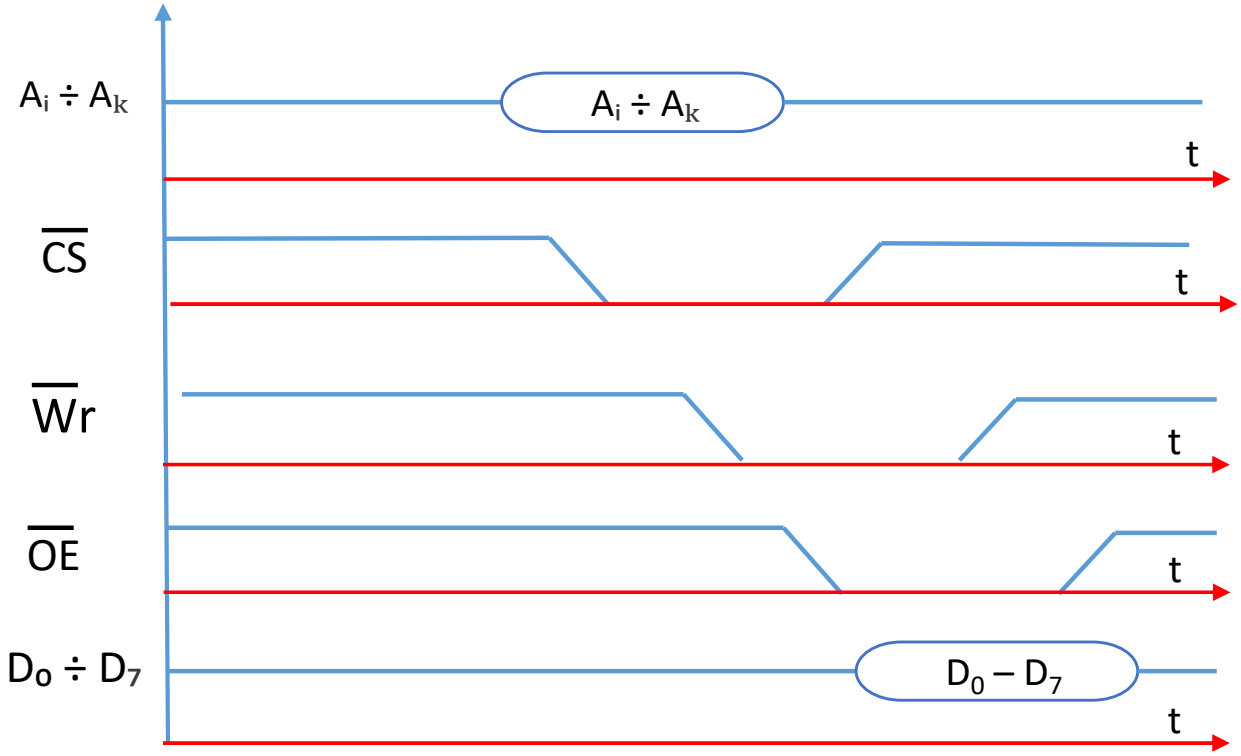


  - Nguyên tắc hoạt động:



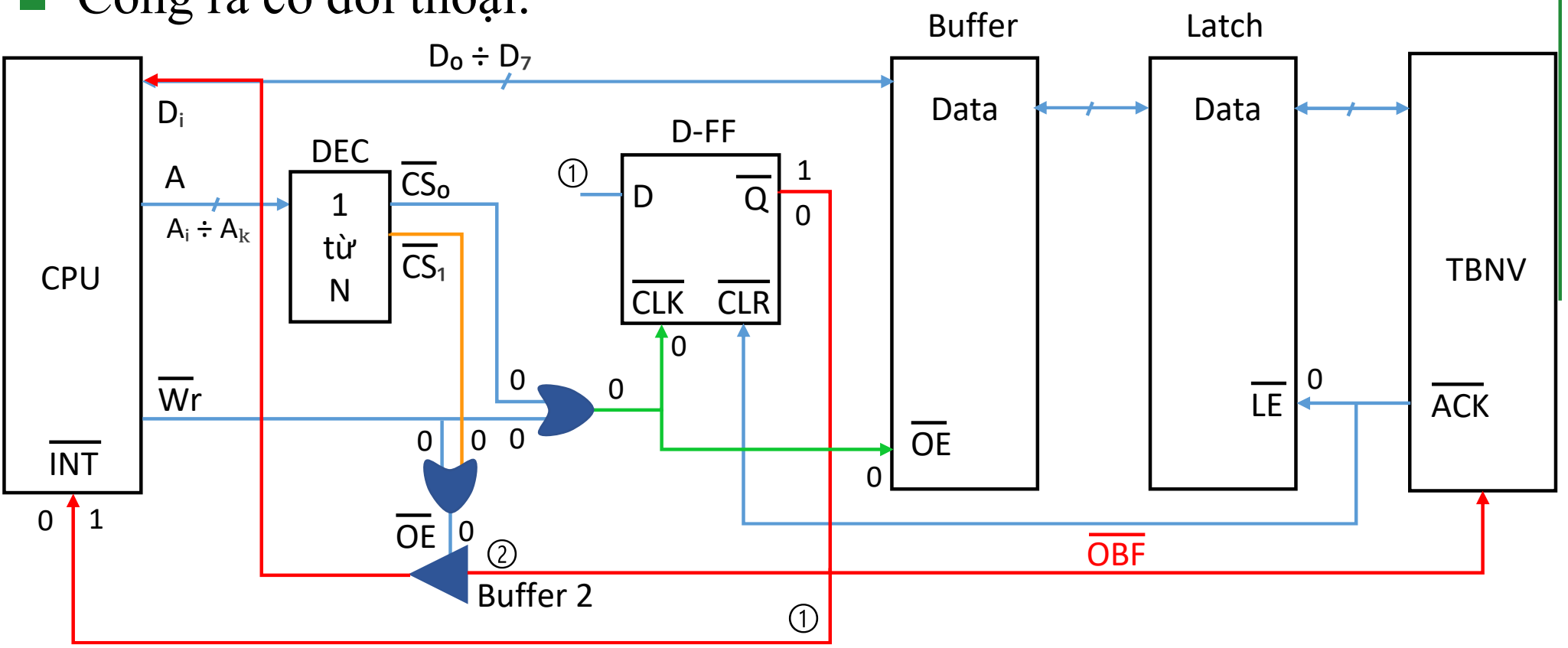
Cổng vào ra

- Giản đồ thời gian:



Cổng vào ra

- Cổng ra có đối thoại:



CPU $\xrightarrow[D_0 \div D_7]{D}$ Buffer

CPU $\xrightarrow[A_i \div A_k]{A}$ DEC $\rightarrow \overline{CS_0} = 0$

CPU $\rightarrow \overline{Wr} = 0$

OR

$\overline{OE} = 0$: Buffer $\xrightarrow{Data}$ Latch

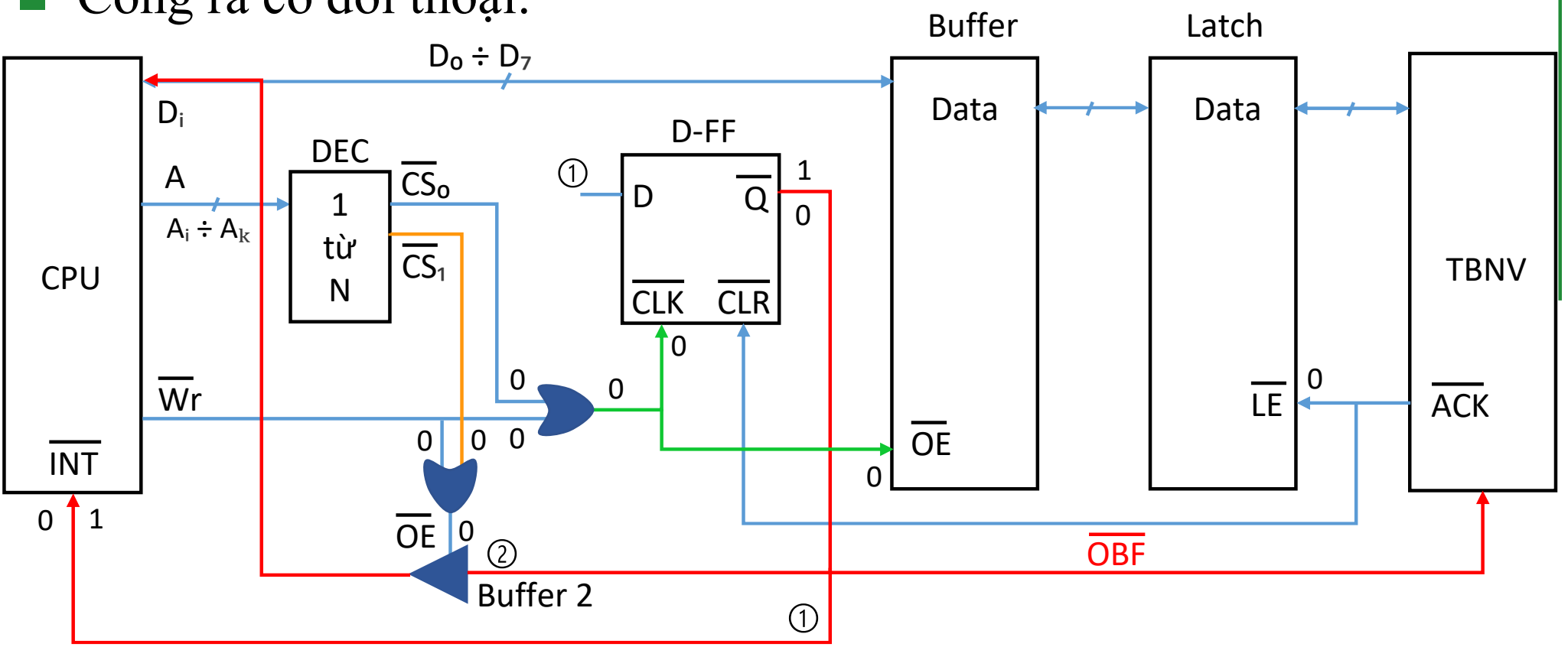
$\overline{CLK} \rightarrow D = 1 \rightarrow \overline{Q} = 0$

D-FF $\xrightarrow[\text{Output Buffer Full}]{\overline{OBF} = 0}$ TBNV

Int CPU ①

Cổng vào ra

- Cổng ra có đối thoại:

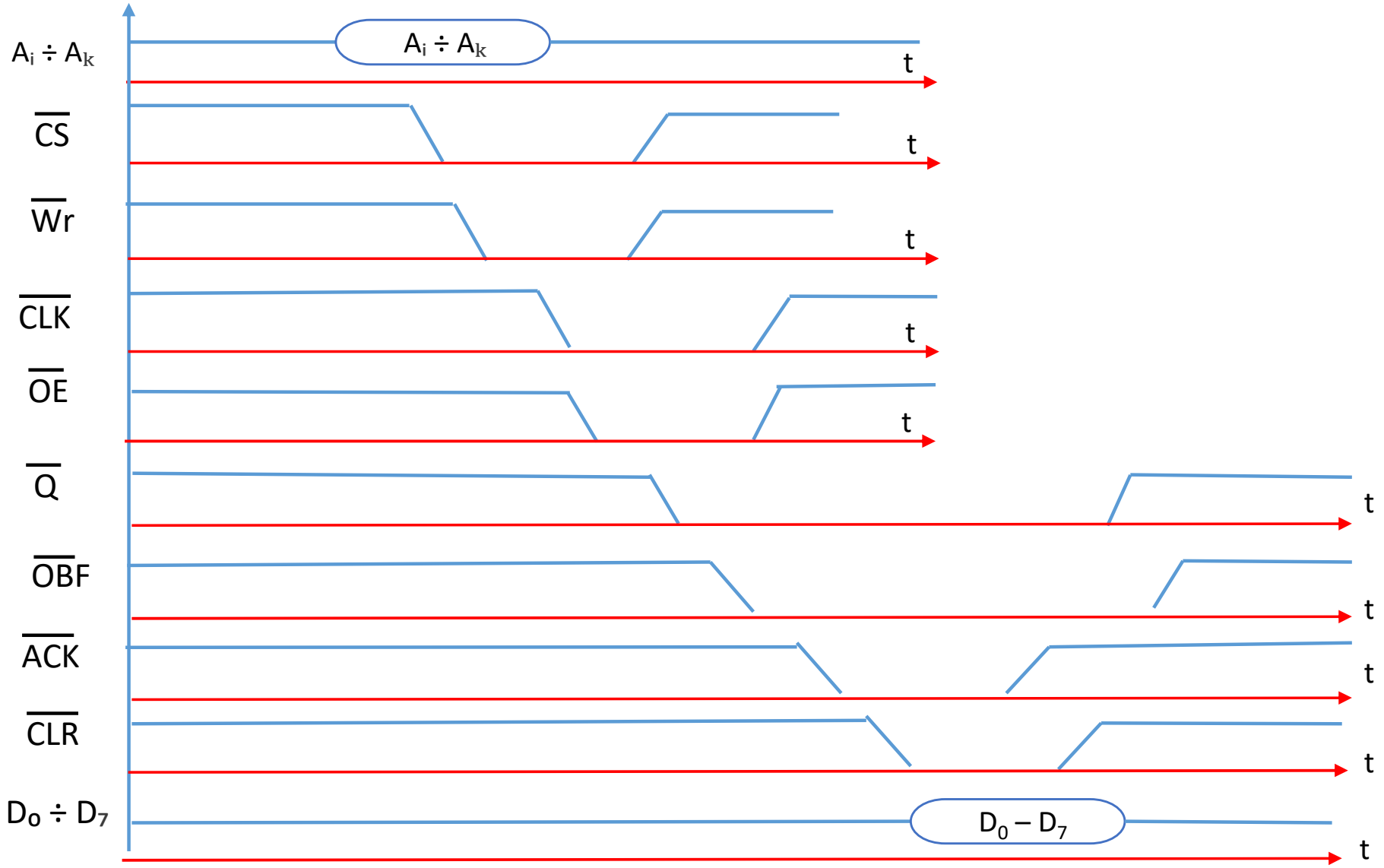


$\overline{ACK} \rightarrow \overline{LE} \rightarrow$ Latch $\xrightarrow{\text{Data}}$ TBNV

$\overline{ACK} \rightarrow \overline{CLR} \rightarrow \overline{Q} = 1 \xrightarrow{\text{TBNV Kthuc đọc}}$ CPU gửi data tiếp

Cổng vào ra

- Giản đồ thời gian:



Cổng vào ra

- Cổng Read back: